

UNITED STATES PATENT AND TRADEMARK OFFICE  
DOCUMENT CLASSIFICATION BARCODE SHEET

---



CATEGORY:

CLEARED

---

ADDRESS  
CONTACT IF FOUND:

10/27/00  
35849 U.S. PTO

10-30-CO

A  
BOX-SEQ

PTO  
35849 U.S. PTO  
10/27/00

Please type a plus sign (+) inside this box → ☐

Approved for use through 10/31/2002 OMB 0651-0032

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

# UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No.	6451.064
First Inventor	Oleg S. Pianykh, et
Title	Radiologist Workstation
Express Mail Label No.	EL282195624US

## APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

- ☒ Fee Transmittal Form (e.g., PTO/SB/17)  
(Submit an original and a duplicate for fee processing)
- ☒ Applicant claims small entity status.  
See 37 CFR 1.27.
- ☒ Specification [Total Pages 52]  
(preferred arrangement set forth below)
  - Descriptive title of the invention
  - Cross Reference to Related Applications
  - Statement Regarding Fed sponsored R & D
  - Reference to sequence listing, a table, or a computer program listing appendix
  - Background of the Invention
  - Brief Summary of the Invention
  - Brief Description of the Drawings (if filed)
  - Detailed Description
  - Claim(s)
  - Abstract of the Disclosure
- ☒ Drawing(s) (35 U.S.C. 113) [Total Sheets 15]
- ☐ Oath or Declaration [Total Pages ]
  - ☐ Newly executed (original or copy)
  - ☐ Copy from a prior application (37 CFR 1.63 (d))  
(for continuation/divisional with Box 17 completed)
    - ☐ **DELETION OF INVENTOR(S)**  
Signed statement attached deleting inventor(s)  
named in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).
- ☐ Application Data Sheet. See 37 CFR 1.76

## ADDRESS TO:

Assistant Commissioner for Patents  
Box Patent Application  
Washington, DC 20231

## ACCOMPANYING APPLICATION PARTS

- ☐ Assignment Papers (cover sheet & document(s))
- ☐ 37 CFR 3.73(b) Statement (when there is an assignee) ☐ Power of Attorney
- ☐ English Translation Document (if applicable)
- ☐ Information Disclosure Statement (IDS)/PTO-1449 ☐ Copies of IDS Citations
- ☐ Preliminary Amendment
- ☐ Return Receipt Postcard (MPEP 503)  
(Should be specifically itemized)
- ☐ Certified Copy of Priority Document(s)  
(if foreign priority is claimed)
- ☐ Other: .....

17. If a CONTINUING APPLICATION, check appropriate box, and supply the requisite information below and in a preliminary amendment, or in an Application Data Sheet under 37 CFR 1.76:

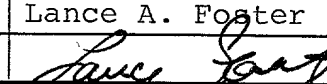
<input type="checkbox"/> Continuation	<input type="checkbox"/> Divisional	<input type="checkbox"/> Continuation-in-part (CIP)	of prior application No. _____/_____
Prior application information:		Examiner _____	Group / Art Unit _____

For CONTINUATION OR DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 5b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

## 18. CORRESPONDENCE ADDRESS

<input type="checkbox"/> Customer Number or Bar Code Label	(Insert Customer No. or Attach bar code label here)	or <input type="checkbox"/> Correspondence address below
--	---	--

Name	Lance A. Foster				
Address	ROY, KIESEL & TUCKER				
	P.O. Box 15928				
City	Baton Rouge	State	Louisiana	Zip Code	70895
Country	USA	Telephone	(225) 927-9908	Fax	(225) 926-2685

Name (Print/Type)	Lance A. Foster	Registration No. (Attorney/Agent)	38,882
Signature		Date	17 OCT 00

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

U.S. PTO  
09/697992  
10/27/00

# FEE TRANSMITTAL for FY 2001

Patent fees are subject to annual revision.

TOTAL AMOUNT OF PAYMENT (\$)  
515.00

## Complete if Known

Application Number  
Filing Date  
First Named Inventor Oleg S. Pianykh, et al  
Examiner Name  
Group Art Unit  
Attorney Docket No. 6451.064

## METHOD OF PAYMENT

1. ☐ The Commissioner is hereby authorized to charge indicated fees and credit any overpayments to:

Deposit Account Number  
Deposit Account Name

☐ Charge Any Additional Fee Required  
Under 37 CFR 1.16 and 1.17

☒ Applicant claims small entity status.  
See 37 CFR 1.27

2. ☒ Payment Enclosed:

☒ Check ☐ Credit card ☐ Money Order ☐ Other

## FEE CALCULATION

## 1. BASIC FILING FEE

Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description	Fee Paid
101 710	201 355	Utility filing fee	355
106 320	206 160	Design filing fee	
107 490	207 245	Plant filing fee	
108 710	208 355	Reissue filing fee	
114 150	214 75	Provisional filing fee	

SUBTOTAL (1) (\$)  
355.00

## 2. EXTRA CLAIM FEES

Total Claims 19  
Independent Claims 7  
Multiple Dependent Claims

Extra Claims Fee from below Fee Paid

20\*\* = 4 X 40 = 160

3\*\* = 4 X 40 = 160

Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description
103 18	203 9	Claims in excess of 20
102 80	202 40	Independent claims in excess of 3
104 270	204 135	Multiple dependent claim, if not paid
109 80	209 40	** Reissue independent claims over original patent
110 18	210 9	** Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) (\$)  
160.00

\*\*or number previously paid, if greater; For Reissues, see above

## FEE CALCULATION (continued)

## 3. ADDITIONAL FEES

Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description	Fee Paid
105 130	205 65	Surcharge - late filing fee or oath	
127 50	227 25	Surcharge - late provisional filing fee or cover sheet	
139 130	139 130	Non-English specification	
147 2,520	147 2,520	For filing a request for ex parte reexamination	
112 920*	112 920*	Requesting publication of SIR prior to Examiner action	
113 1,840*	113 1,840*	Requesting publication of SIR after Examiner action	
115 110	215 55	Extension for reply within first month	
116 390	216 195	Extension for reply within second month	
117 890	217 445	Extension for reply within third month	
118 1,390	218 695	Extension for reply within fourth month	
128 1,890	228 945	Extension for reply within fifth month	
119 310	219 155	Notice of Appeal	
120 310	220 155	Filing a brief in support of an appeal	
121 270	221 135	Request for oral hearing	
138 1,510	138 1,510	Petition to institute a public use proceeding	
140 110	240 55	Petition to revive - unavoidable	
141 1,240	241 620	Petition to revive - unintentional	
142 1,240	242 620	Utility issue fee (or reissue)	
143 440	243 220	Design issue fee	
144 600	244 300	Plant issue fee	
122 130	122 130	Petitions to the Commissioner	
123 50	123 50	Petitions related to provisional applications	
126 240	126 240	Submission of Information Disclosure Stmt	
581 40	581 40	Recording each patent assignment per property (times number of properties)	
146 710	246 355	Filing a submission after final rejection (37 CFR § 1.129(a))	
149 710	249 355	For each additional invention to be examined (37 CFR § 1.129(b))	
179 710	279 355	Request for Continued Examination (RCE)	
169 900	169 900	Request for expedited examination of a design application	

Other fee (specify)

\* Reduced by Basic Filing Fee Paid

SUBTOTAL (3) (\$)

## SUBMITTED BY

Name (Print/Type) Lance A. Foster  
Signature  
Registration No (Attorney/Agent) 38,882  
Telephone (225) 927-9908  
Date 27 Oct 2000

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, DC 20231.

**COMPACT DISC  
COMPUTER PROGRAM LISTING  
TRANSMITTAL**

INVENTOR: Pianykh, et al.

TITLE: Radiologist Workstation

ATTY. DOC. # 6451.64

Machine Format                      IBM-PC

Operating System                  MS-Windows

<u>Name</u>	<u>Size</u>	<u>Date Last Modified</u>
Dicom_h	37.4 kB	10/27/00
DICOMLib	2.05 MB	10/27/00
GUILib	58.3 kB	10/27/00
hlp	188 kB	10/27/00
Image	81.6 kB	10/27/00
Query	139 kB	10/27/00
RayTraceDialog	381 kB	10/27/00
res	222 kB	10/27/00
Tools	6.65 kB	10/27/00
WinMetheusLib	199 kB	10/27/00
Zlib113	135 kB	10/27/00



**Separator Sheet**



**AAA**

**Application Text**

**Place Papers behind this sheet in the following order:**

1. Preliminary Amendment
2. Specification (English Language Only)
3. Claims
4. Abstract

**1**

[illegible]

**Invented By:**

**Oleg S. Panykh, David Troendle, John M. Tyler, and Wilfrido Castaneda-Zuniga**

**Attorney Docket No. 6451.064**

Express Mail # EL 282195624US

Appendix 1 is a printout of the source code, which is incorporated by reference herein, executing the functions described in the accompanying specification. Appendix 2 is a computer program listing appendix consisting of two compact discs, each having a complete copy of the source code executing the functions described in the accompanying specification.

5 The contents of the compact discs are incorporated by reference herein. High level folders on the compact discs are:

<u>Name</u>	<u>Size</u>	<u>Date Last Modified</u>
Dicom_h	37.4 kB	10/27/00
10 DICOMLib	2.05 MB	10/27/00
GUILib	58.3 kB	10/27/00
hlp	188 kB	10/27/00
Image	81.6 kB	10/27/00
Query	139 kB	10/27/00
15 RayTraceDialog	381 kB	10/27/00
res	222 kB	10/27/00
Tools	6.65 kB	10/27/00
WinMetheusLib	199 kB	10/27/00
20 Zlib113	135 kB	10/27/00

## **BACKGROUND OF INVENTION**

The present invention relates to improvements in a Picture Archiving and Communications System (PACS). In particular, the invention relates to a set of software tools, which will create a client-server PACS significantly enhancing the use, transfer, and analysis of  
25 medical images stored in a medical image database.

PACS operating upon medical image databases almost universally comply with the Digital Imaging in COmmunication and Medicine (DICOM) standard. DICOM is the widely accepted standard for digital medical data representation, encoding, storage and networking. The DICOM standard is specified in 14 volumes (traditionally numbered as PS3.1 – PS3.14),  
30 available from National Electrical Manufacturers Association (NEMA) and is well known to

those working in the digital medical imaging area. The standard provides strict guidelines for medical digital imaging, and is supported by virtually all present medical system manufacturers. The DICOM standard is object-oriented and represents each information entity as a set of Data Elements (such as Patient Name, Image Width) with their respective values encoded with DICOM Dictionary (PS3.5-PS3.6). While there are numerous existing PACS with many specialized features, these PACS could be enhanced with new forms of data compression and other features rendering the PACS more efficient to the user.

Existing PACS have included methods of lossless compression, such as the well-known Joint Photographer Experts Group (JPEG) algorithm. Lossless compression is naturally a more limited form of compression since it is necessary to preserve all information in the image. Because the degree of compression is more limited, there is always a need in the art to provide more efficient methods of losslessly compressing image data. Therefore, the present invention provides an improved method of lossless compression.

In addition to JPEG lossless compression, DICOM provides a method of JPEG lossy compression. DICOM support for the standard lossy JPEG compression can be implemented with an object-oriented JPEG class having the following *Compress* method:

```
int JPEG::Compress(BYTE* input, BYTE* output, int quality, int bpp=1, bool  
rgb=false)
```

wherein the parameters of this method are:

*BYTE\* input* – a pointer to the original image pixel array,  
*BYTE\* output* – a pointer to the compressed image pixel array  
*int quality* – compression quality  
*int bpp* – bytes per pixel in the original image  
*int rgb* – color flag of the original image (*true* if a color image is given, and *false* for grayscale).

The function "*Compress*" returns an *int* or integer value which is the size of the compressed image (i.e. the size (number of bytes) of the *output* array). Naturally, in lossy compression, the amount of compression will correspond to a certain amount of information loss in the original image. In standard JPEG, the amount of information loss in the *Compress* function is specified with JPEG *quality* parameter, which ranges from 0-100%. A *quality*=100% corresponds to minimal loss and low compression ratios, while a smaller *quality* percentage value corresponds to a higher loss and a higher compression ratio. The choice of the *quality* value permits the user to compromise between the perceptual image quality and the compressed image size. However, the main disadvantage of the *quality* parameter is that it is not directly related to the corresponding compression ratio: i.e., setting quality to 50% will produce different compression ratios for different images. For example, at 50% quality, a low entropy (low information content) image would be compressed much more than a high entropy (high information content) image. Therefore, if one is attempting to compress images with varying information content to the same predefined compression ratio, the *quality* parameter becomes practically useless. Thus, what is needed in the art and what is provided in one embodiment of the invention described below, is a new JPEG compression method based on a predefined compression ratio, rather than compression quality.

PACS are often implemented on network systems such as Local Area Networks (LANs) and include a server system controlling the transfer of medical image data from storage devices to multiple client systems. Since medical image files are typically large data files, it is not uncommon for simultaneous data requests from multiple client systems to heavily burden the existing bandwidth of the network's data link. Insufficient network bandwidth to accommodate the demand of client systems results in undesirably slow download of data to the client systems.

Network bandwidth on any given network can also vary significantly depending on the number of clients currently using the network or other factors placing computational load on the system.

5        Compressing an image prior to it being transferred across the network is one manner of compensating for insufficient bandwidth. Compressing data on a local computer is typically done much faster than downloading an image from a remote system. Therefore, compressing data by  $n$  times before transmission will reduce the download time of that data by nearly  $n$  times. As mentioned, the DICOM standard already includes JPEG lossless and lossy compression for local image storage. It may often be possible to alleviate most bandwidth  
10        problems by compressing data to a large degree (typically through lossy compression). However, lossy compression results in the loss of some information in the compressed data and this loss may be unacceptable in medical images which are used in certain applications. Even where lossy compression is acceptable, it is desirable to minimize the information loss.

15        What is needed in the art is a method of maintaining an acceptable download time across a network while compressing image data only to the degree necessary to maintain that download time. In other words, a method of adjustably compressing data in response to the available network bandwidth.

20        Another shortcoming of existing PACS is their failure to encapsulate audio information in DICOM objects. Audio information is often used in medical practices to record reports, clinical sounds, and the like. However, unlike image data, the DICOM standard provides no support for sound data encapsulation in DICOM objects. Typically, DICOM compliant systems store and interpret audio data separately, with the respective DICOM image object containing a pointer to the present audio data location. The main deficiency of this approach is

that the audio data is separated from the rest of the DICOM information. Thus, the audio data can be lost, it is not available at the same time as the image, and its retrieval involves additional processing steps. However, the generality of the DICOM standard allows representation of any information as a part of a DICOM object. Therefore, it would be a significant advancement in the art to provide a method for encapsulating audio information in a DICOM object.

Ray Tracing is another software tool commonly used in medical imaging. Ray tracing is a method of analyzing three-dimensional data in order to produce a properly shaded three dimension appearing image on a monitor. As described in more detail below, ray tracing performs two basic steps, ray generation and data volume traversal, for all of (or at least a large number of) the pixels on the monitor displaying the image. This repetitive computational processing thus requires unusually high computing power and consumes an undesirable amount of processing time. Improving the speed at which ray generation and data volume traversal are performed would significantly speed up ray tracing image production because of the repetitive nature of these two operations.

## **SUMMARY OF THE INVENTION**

The present invention provides a method of lossless image compression comprising the steps of (a) taking a sample of pixel neighborhoods from an image file, (b) using the sample of pixel neighborhoods to determine a series of predictive coefficients values related to the pixel neighborhoods, (c) determining prediction residual values based on the coefficients values, and (d) losslessly compressing the prediction residual values.

The invention also provides a method of adjusting the quality parameter in a quality controlled compression routine in order to compress image data a predetermined compression ratio. The method comprises the steps of: (a) receiving a desired compression ratio, (b)

selecting an estimated quality value, (c) compressing the image data based on the quality value and calculating an intermediate compression ratio from the compressed image data, (d) adjusting the quality value in iterative steps and recalculating the intermediate compression ratio, (e) returning a final quality value after a predetermined time period or when the predefined ratio is achieved, and (f) compressing the image data to the final quality value using the quality controlled compression routine.

The invention also includes a method of adjusting the compression ratio of image data based upon the available bandwidth of a network link across which the image is transmitted. The method comprises the steps of: (a) determining the size of an image to be transmitted, (b) selecting a desired download time for downloading the image from a network link, (c) determining a current download time based upon current network conditions, and (d) compressing the image if the current download time is greater than the desired download time.

The invention also includes a method of encapsulating audio data in a DICOM object and selectively retrieving the data. The method comprises the steps of: (a) providing a DICOM compliant recording function having parameters representing a recording time and a record size of the audio data, and (b) providing a DICOM compliant playback function having a parameter representing a playback start position.

The invention also includes an improved integer based ray tracing method for constructing two-dimensional images from three-dimensional data. The ray tracing method comprises the steps of: (a) receiving a set of three-dimensional image data containing an object of interest, (b) receiving an observer position, (c) establishing a ray angle from the observer position, through the three-dimensional data, to a two-dimensional image plane, (d) adjusting the ray angle such that the directional components of the ray angle are rational numbers, (e)



back projecting from a selected number of picture elements on the two dimensional image plane a series of rays parallel to the ray angle, the rays passing through the three-dimensional image data to origin points, (f) for each ray intersecting the object of interest, determining a relative distance between the origin point and a point of contact where a ray intersects the object of interest, (g) determining a surface angle relative to the origin point for each of the points of contact, and (h) adjusting the intensity of the picture elements on the two dimensional image plane relative to the surface angle of the points of contact.

The invention also includes a method of reducing flicker caused by a magnifying window moving across an image on the display screen. The method comprises the steps of: (a) storing the image in the memory, (b) storing a first window position in the memory, (c) reading a second window position, which overlaps the first window position, (d) determining what portion of the first window position is not covered by the new window position, and (e) restoring from memory that portion of the image which corresponds to the portion of the first window not covered by the second window.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

Figure 1 is an illustration of a pixel neighborhood.

Figure 2 is a pseudocode segment illustrating the predictive lossless compression method of the present invention.

Figure 3 is a pseudocode segment illustrating the ratio based lossy compression method of the present invention.

Figure 4 is a pseudocode segment illustrating the method for determining current network bandwidth for application entities.

Figure 5 is a pseudocode segment illustrating the method for determining if compression is necessary and to what degree compression should be carried out.

Figure 6 is a pseudocode segment illustrating a timing method employed in the present invention.

5        Figure 7 is a pseudocode segment illustrating a sound recording function employed in the present invention.

Figure 8 is a pseudocode segment illustrating the method for implementing the SoundRecorder object of the present invention.

Figure 9 is a conceptual representation of a three-dimensional data set.

10        Figure 10 is a schematic representation of prior art ray tracing methods.

Figure 11 is a detail of multiple rays striking the surface of an object of interest.

Figure 12 is an graphical illustration of a prior art method used to determine the surface angle orientation of the object of interest at the point of ray impact.

15        Figure 13(a) is a graphical representation of a three-dimensional data set being traversed by a ray.

Figure 13(b) is a graphical representation of an individual cell within a three-dimensional data set being traversed by a ray.

Figure 14 is a graphical illustration of the ray tracing method of the present invention.

20        Figure 15(a) is a graphical representation of a three-dimensional data set being traversed by a ray in the method of the present invention.

Figure 15(b) is a graphical representation of an individual cell within a three-dimensional data set being traversed by a ray in the method of the present invention.

Figure 16 is a graphical illustration of the parabolic interpolation method of the present invention.

Figure 17 is a graphical illustration of the known pixel values used to interpolate the unknown pixel values in the present invention.

Figure 18 is a graph illustrating an optimum step size  $d$  when using the race tracing method of the present invention.

Figure 19 is an illustration of an image on a view screen with a magnified window appearing on the screen.

Figures 20(a)-20(c) illustrates how a prior art magnifying window proceeds through successive steps to update the window position.

Figure 21 illustrates the improved method of updating the magnifying window in accordance with the present invention.

Figure 22 is a flow chart of the improved method of updating the magnifying window in accordance with the present invention.

Figures 23(a)-23(c) illustrate methods of optimizing contrast in the magnified image of the present invention.

## **DETAILED DESCRIPTION**

Where appropriate, the terminology used in the following description is a part of the DICOM standard and is well known to persons of ordinary skill in the medical imaging art. For the embodiments described herein, the definitions of various DICOM classes and functions are presented in C++ object-oriented syntax and the functional interface of these classes is illustrated in the function definitions and pseudocode appearing herein.

### **Enhanced (predictive) lossless compression.**

One aspect of the present invention is to provide an improved lossless compression method. Lossless compression is the compression of data without any informational loss – i.e., the original data (for example an image) can be completely recovered from its compressed version. One advantageous method of implementing a lossless compression method for images is to use natural inter-pixel correlation, i.e. the tendency for neighboring pixels in an image to have close numerical values. If an  $N \times M$  digitized image is represented as an  $N \times M$  matrix  $I(x,y)$ , where the value of  $I(x,y)$  represents pixel's intensity at the point  $(x,y)$ , a predictive model for each pixel's intensity can be written as:

$$I(x,y) = \sum_{0 \leq i,j \leq a} c_{i,j} I(x-i,y-j) + e(x,y), \quad (1)$$

where  $c_{i,j}$  are coefficients, and  $e(x,y)$  is the prediction residual.

Figure 1 illustrates a set of pixels, which may be referred to as a pixel neighborhood. The left-bottom neighborhood adjacent to the reference pixel  $(x,y)$  will be considered the  $I(x-i,y-j)$  neighborhood where  $i$  and  $j$  are equal to the neighborhood size  $a$ . In Figure 1,  $a$  is equal to 3.

If  $c_{i,j}$  are known, equation (1) illustrates how  $I(x,y)$  can be losslessly recovered from its left-bottom pixel neighborhood  $I(x-i,y-j)$ , and a small error  $e(x,y)$ . Thus, the only information needed to represent the compressed image is the values  $e(x,y)$  and the coefficients  $c_{i,j}$ . Since the values of  $e(x,y)$  can usually be efficiently and losslessly encoded with entropy-based compression (such as Huffman compression), a significant degree of compression of the original image may be realized using the method of the present invention.

Modern lossless compression techniques such as JPEG and JPEG-LS use predefined coefficient values and relatively small neighborhoods of  $a=1$  or  $a=2$  pixels. However, it has been discovered that extending the neighborhood size (increasing the  $a$  value) to  $a=3$  and  $a=4$ ,

and choosing  $c_{ij}$  based on the given image produces substantially better compression ratios. These superior compression ratios are on the order of a 20%-30% increase over JPEG techniques. This is a significant improvement when dealing with lossless compression.

The compression technique of the present invention generally comprises four steps. First, the image will be sampled for a series of pixel neighborhoods from which the prediction coefficients  $c_{ij}$  will be calculated. Second, the coefficients  $c_{ij}$  will be determined by linear regression. Third, equation (1) will be used to compute  $e(x,y)$  for each pixel. Lastly, the values for  $e(x,y)$  will be losslessly compressed.

#### Sampling the image for pixel neighborhoods.

In order to compute  $c_{ij}$  in equation (1), a population of pixel neighborhoods must be chosen. Naturally, the minimum number of independent samples required must be at least the number of coefficients sought to be calculated. While the maximum number of samples could equal the number of pixels in the entire image, this sampling strategy would become excessively time-consuming. A preferred alternative is to choose a set of sampling neighborhoods  $(x_s, y_s)$ , where  $S$  is the total number of samples and  $s$  represents a particular sampling index ranging from 1 to  $S$ . The set of  $S$  samples may be obtained by sampling the image in a uniform manner using a predetermined sampling step  $d$ . Thus, it will be readily apparent how the following pseudocode would operate to select a set of samples:

```

s=1;
for(x=1; x<N; x+=d)
{
    for(y=1; y<M; y+=d)
    {
        Consider the current point (x,y) as the right-top point of
        the next neighborhood sample # s;
        Increment s=s+1;
    }
}

```

This routine will produce  $S=NM/d^2$  samples with each sample being an  $[(a+1) \times (a+1)]-1$  set of adjacent pixels from the original image. As suggested above, the value of  $d$  must be chosen to insure a sufficient number of samples to solve for the number of coefficients  $c_{ij}$  to be computed from equation (1). This requires that  $S$  obey the inequality

$$S = NM / d^2 > (a+1)^2, \quad (2)$$

which in turn leads to the constraint

$$d < d_0 = \sqrt{NM} / (a+1). \quad (3)$$

For practical purposes, and with large images, one may choose smaller values of  $d$  (i.e. more neighborhood samples) to insure a more accurate approximation of  $c_{ij}$  from the samples applied to equation (1). One preferred embodiment of the invention uses a sampling step  $d$  governed by the relationship  $d=\max(1, d_0/10)$ .

#### Finding $c_{ij}$ with linear regression

Equation (1), taken over all samples  $s$ , results in a typical linear regression problem that can be resolved with the well-known least-squares approach. The coefficient values  $c_{0,1}$ ,  $c_{1,0}$ ,  $c_{1,1}$ , ...,  $c_{a,a}$  may be chosen to minimize the sum of squared errors  $e(x_s, y_s)$  in equation (1) over all the neighborhood samples. The steps of solving for the coefficients  $c_{ij}$  may be best visualized using matrices with the following notations:

Vector  $C = [c_{0,1}, c_{1,0}, c_{1,1}, \dots, c_{a,a}]$  – unknown prediction coefficients  
 Vector  $V = [I(x_1, y_1), I(x_2, y_2), \dots, I(x_s, y_s)]^T$  – left-hand parts of (1) for each sample  $s$ .  
 Vector  $W_{ij} = [I(x_1-i, y_1-j), I(x_2-i, y_2-j), \dots, I(x_s-i, y_s-j)]^T$  – right-hand parts of (1) for each sample  $s$  and each  $(i,j)$  pair.

Equation (4) illustrates how equation (1) may be rewritten as the matrices

$V = C[W_{01}, W_{1,0}, W_{1,1}, \dots, W_{a,a}] = CW$ , and, after multiplication by  $W^T$ , may be written as

$$VW^T = C(WW^T) = CF \quad (4)$$

after which  $C$  may be solve for as

$$C = (VW^T)(WW^T)^{-1} = (VW^T)(F)^{-1} \quad (5)$$

Matrix  $F$  has  $f \times f$  coefficients  $F_{p,q}$ , where  $f=(a+1)^2-1$ . Computing  $C$  has two basic steps. First, the inversion of this matrix  $F$  is determined. Once  $F^{-1}$  is found, coefficients  $C$  may be  
5 determined from equation (5). Typically, the calculations may be simplified by quantizing the coefficients  $c_{ij}$  into an integer form represented by  $k_{ij}$ . This may be accomplished by multiplying  $c_{ij}$  by  $2^{10}$  or 1024. Thus,  $k_{ij}=[1024 * c_{ij}]$  or  $c_{ij}=k_{ij}/1024$ , or in matrix notation,  $C=K/1024$ .

Most practically, the step of actually determining  $F^{-1}$  will not be carried out, but instead  
10 equation (4) will be solved using Integer Gaussian elimination. As above,  $C$  is first quantized by multiplying both sides in equation (4) by 1024:

$$(1024 * VW^T) = 1024 * CF = KF$$

and solving

$$(1024 * VW^T) = KF$$

15 for  $K$  with traditional Gaussian elimination techniques well known in the art. One suitable software routine for Gaussian elimination is found in "Numerical Recipes in C", 2nd Ed., Press, Teukolosky, Vetterling, and Flannery, Cambridge University Press, 1992, the relevant portions of which are incorporated herein by reference. All vectors in the equation  
(1024 \*  $VW^T$ ) =  $KF$  have integer components, which ensures very fast convergence. A good  
20 initial approximation of  $K$  may be  $K=[512, 512, 0, 0, \dots, 0]$  (corresponding to  $C=[0.5, 0.5, 0, 0, \dots, 0]$ ). It is noted that these values for  $C$  are the fixed values used in JPEG compression. Therefore, all iterations of Gaussian elimination beyond this initial estimate of  $C$  insure that the resulting coefficients will produce compression ratios superior to JPEG.

Those skilled in the art will recognize it may also be practical to find the  $F^{-1}$  coefficients from direct inversion of the  $F$  matrix is small (i.e.  $f < 4$ ). For example, if  $f=2$ ,  $F^{-1}$  would be found as:

$$\begin{pmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{pmatrix}^{-1} = \begin{pmatrix} f_{22}/r & -f_{12}/r \\ -f_{21}/r & f_{11}/r \end{pmatrix}, r = f_{11}f_{22} - f_{12}f_{21}.$$

5 While this approach can be used efficiently for  $f < 4$ . Once  $f$  exceeds 3, it is more practical to use the Gaussian elimination method described above.

Applying equation (1) to compute  $e(x,y)$ .

Once coefficients  $c_{i,j} = k_{i,j}/1024$  are determined, the most efficient manner of determining  $e(x,y)$  is to utilize the quantized equivalent of equation (1):

10 
$$I(x,y) = \left( \sum_{0 \leq i \leq a, 0 \leq j \leq a} k_{i,j} I(x-i, y-j) \right) >> 10 + e(x,y), \quad (6)$$

where  $>>$  stands for a 10-bit right shift operation, which is equivalent to division by 1024, but computationally faster in binary systems than standard division operations. The pseudocode seen in Figure 2 implements the predictive lossless image compression in a manner readily followed by a person of ordinary skill in the programming arts. Note comments indicated by  
15  $"/"$ .

Losslessly compressing and decompressing  $e(x,y)$

Once the integer sequence for  $e(x,y)$  has been calculated as above, the  $e(x,y)$  sequence will be encoded with any entropy-based coding scheme, such as Huffman compression. The encoded  $e(x,y)$  sequence will form the basis of the compressed image file. Additionally, the  
20 sequence of  $k_{i,j}$  is stored as a header of the compressed image.



It will be obvious that decompressing the image file is substantially the reverse of the above described steps. To decompress the image file, one first reads the sequence of  $k_{i,j}$  coefficients from the compressed image file header. Then the remaining sequence of  $e(x,y)$  is then decoded with the entropy based scheme (e.g. Huffman decoding). Finally, the original image  $I(x,y)$  may be recovered from  $e(x,y)$  and  $k_{i,j}$  using equation (6) and a procedure substantially the reverse of that seen in the pseudocode.

**Real-Time JPEG.** Another aspect of the present invention is providing an improved JPEG *Compress* function. The *Compress* function should be capable of compressing an image to a predefined compression ratio. This compression ratio-based JPEG is implemented similarly to traditional JPEG, but takes a compression ratio parameter (*ratio*) instead of the *quality* parameter. The parameters other than *ratio* are the same as described above for traditional JPEG *Compress* function. In the description below, the new function is denominated *CompressR*:

```
int RJPEG::CompressR(BYTE* input, BYTE* output, double ratio, int bpp=1, bool  
rgb=false).
```

Thus, when a compression ratio is provided, the function will return an integer value that is the original image size compressed by the given ratio. For example, *ratio*=2.0 means that the original image must be reduced in size by at least one half of its original size.

The present invention also includes a further improvement upon the standard JPEG *Compress* function. This second modification includes also a time constraint limiting the time during which the function may operate to achieve the desired compression ratio.

```
int RTJPEG::CompressRT(BYTE* input, BYTE* output, double ratio, double maxtime,  
int bpp=1, bool rgb=false),
```

where *maxtime* gives the maximum time in seconds in which the routine is allowed to perform the compression. Of course, given *CompressRT* function, the previous, *CompressR* function can be always expressed as

```
5      CompressR(BYTE* input, BYTE* output, double ratio, int bpp=1, bool rgb=false)
      {
          return CompressRT(input, output, ratio, 1000000, bpp, rgb);
      }
```

10 where some very large *maxtime* value (e.g. 1,000,000) is used to indicate that the time constraint is not required.

One manner in which the *CompressRT* function may be implemented is by using the standard JPEG *Compress* with an appropriate *quality* value estimated iteratively to ensure the proposed compression ratio. One example of implementing the *CompressRT* function in a software routine may be illustrated with the pseudo code seen in Figure 3.

15 It will be understood that in the pseudo code of Figure 3, comment lines are preceded by the *"/"* symbol. Those skilled in the art will recognize many standard functions presented in C++-type pseudocode; for example the *clock()* function discussed below. The other functions like *Size()* can be easily implemented by those skilled in the art. These standard functions are well known in the software programming art and need not be elaborated upon further.

20 As indicated above, the *CompressRT* function will take the parameters *ratio* and *maxtime* and return an integer value equal to the size of the compressed image. Viewing Figure 3, the first step in the *CompressRT* routine is to determine the size of the original image with a *Size()* function. Next, minimum and maximum quality values (*q0* and *q1*) are set. For illustration purposes only, these values have been set to 5 and 95 respectively. While (*q0* and  
25 *q1*) could be initially set to any percentages, 5 and 95 are good initial values and selecting a narrower range should not materially enhance computational speed. Using the standard JPEG

*Compress* function with  $q0$  and  $q1$ , respective compression ratios ( $r0$  and  $r1$ ) corresponding to the minimum and maximum quality values may be determined. This is accomplished by dividing the size of the original image by the size of the image compressed to the qualities  $q0$  and  $q1$ . It will be understood that the lowest quality value results in the highest compression ratio  $r0$ . Similarly, the highest quality value results in the lowest compression ratio  $r1$ .

The routine determines whether the desired or predetermined *ratio* is either lower than the lowest compression ratio ( $r1$ ) or higher than the highest compression ratio ( $r0$ ). If the first condition of  $ratio < r1$  is satisfied, then even the most quality-preserving compression at  $q=q1=95\%$  already achieves ratios higher than the desired *ratio*, therefore the image is compressed to this level, and the *CompressRT* function exits. If  $ratio > r0$  is true, then lossy JPEG can provide at most a  $r0$  compression ratio. Since the compression ratio *ratio* is impossible to achieve, the image is compressed to the highest possible  $r0$  level, and the *CompressRT* function exits.

However, assuming the parameter *ratio* is within  $r0$  and  $r1$ , the routine will begin the steps required to estimate a quality  $q$  which corresponds with the desired compression ratio, *ratio*. Two variables, *start* and *finish* of the type *clock\_t*, will be declared. *Start* will be initialized to the present clock value by a standard function such as *clock()*. A variable *duration* will be initialized to zero and a variable  $r$  will be initialized to the value of *ratio*.  $r$  will serve as an intermediate value of the compression ratio while the routine performs the iterative steps governed by the condition *while*(*duration* < *maxtime*). The first step in the *while* loop is the estimate of the quality value  $q$ . This estimate may be made by any number of methods, with two preferred methods being shown in the pseudo code. The linear estimate uses the parameters  $q$ ,  $q0$ ,  $q1$ ,  $r$ ,  $r0$ , and  $r1$  in the equation  $q = q0 + (r - r0) * (q1 - q0) / (r1 - r0)$ .

Alternatively, a simpler bi-section estimate may be obtain by the equation  $q = (q1+q0)/2$ . Obtaining an estimate of  $q$  allows an updated intermediate compression ratio  $r$  to be found by dividing the size of the original image by the size of the image compressed to the intermediate quality  $q$ .

5        The intermediate compression  $r$  is then compared to the desired compression, *ratio*. If  $r$  is less than *ratio*, the lowest ratio  $r1$  is set equal to  $r$ . If  $q1$  is equal to  $q$ , then the routine will exit the *if* loop, otherwise  $q1$  will be set equal to  $q$  before another iteration is performed within the *if* loop. Alternatively, if  $r$  is greater than *ratio*, the highest ratio  $r0$  is set equal to  $r$ . If  $q0$  is equal to  $q$ , then the routine will exit the *if* loop, otherwise  $q0$  will be set equal to  $q$  before  
10    another iteration is performed within the *if* loop. It will be understood that this loop provides an iterative process where the changing integer values of  $q1$  and  $q0$  will converge upon the intermediate value of  $q$ . In fact, it can be derived mathematically that the bisection iterative process will take at most  $\lceil \log_2(95-5) \rceil = 7$  iterative steps to converge.

      The point of convergence will provide a quality value integer  $q$  which produces the  
15    compression value  $r$  closest to the desired compression value, *ratio*. It should be noted that after each iteration, the variable *duration* is recalculated by subtracting the *start* variable from the *finish* variable and dividing by the clock frequency (i.e. *CLOCKS\_PER\_SEC*). If the *duration* of the iterative process exceeds *maxtime*, the iterative loop will be exited with the best compression ratio  $r$  and quality  $q$  estimates found up to this point in the program's execution.

20        Once the iterative loop is exited, the last intermediate quality value  $q$  will be passed to the standard JPEG *Compress* function. This value of  $q$  will be used to compress the input image and the size of the input image will be returned to the main function *RTCompress*. In this

manner, the function *RTCompress* determines the size of an input image compressed to a desired compression value, *ratio*.

This version of *CompressRT* just described uses the entire image to estimate a compression quality sufficient to achieve the desired compression ratio. This is practical when the estimation process on local computer (i.e. server) can be run fast enough not to interfere with other operations (such as transferring images across a network). If greater computational speed is required, then a smaller sub-sample of the original image may be used to estimate the ratio and quality with *CompressRT*, and then the estimated quality value may be used to compress the original, full-sized image.

The following example illustrates how this sub-sample method reduces computational time. In the above code, each step requires one *Compress()* call to update the current estimated compression ratio *r*, and two more calls are made with *q0=5* and *q1=95* quality values before the iterations start. That means that with the worst case scenario (using the bisection estimate method), *CompressRT()* will take approximately  $7+1+1=9$  times longer to execute compared to standard JPEG *Compress()*. When network bandwidth is low, this computational time overhead will have negligible effect on the total image download time. However, where bandwidth is high and download time comparatively fast, the computational overhead may be reduced by using a smaller image sub-sample to estimate the ratio in *CompressRT()*. One suitable technique is to use the central part of the original image, with  $1/7$  of the original image size in each dimension. In this case, the number of pixels in this sample will be  $1/7 * 1/7 = 1/49$  of that of the original image size. Since JPEG compression complexity is proportional to the number of pixels operated upon, 9 iterations for  $1/49$  of the original data will take  $9*(1/49) = 0.18$ —reducing time overhead to at most only 18% of the full image JPEG compression time.

**Dynamic DICOM Image Compression.** Another embodiment of the present invention provides a method of adjustably compressing data in response to the available network bandwidth. This method will be referred to herein as "dynamic compression". Dynamic image compression can be implemented on top of the DICOM standard as a set of classes/services fully compliant with DICOM object-oriented design. Where appropriate, the terminology used in the following description is a part of the DICOM standard. The definitions of dynamic DICOM compression classes are presented in C++ object-oriented syntax and the functional interface of these classes is illustrated in the pseudocode reference in connection therewith.

#### 1) DICOMObject Class

The DICOMObject class represents a DICOM *Data Set* defined in the part PS3.5, Section 7 of the DICOM standard. The DICOMObject class will support the following functions which are utilized in the dynamic compression program of the present invention:

*int - GetSize()* - This function returns the size of a DICOM object in bytes. The returned size is defined as the total size of all DICOM data element values and data element tags, as specified in part PS3.5 of the DICOM standard.

*int Compress(bool lossless, double ratio, double maxtime=1000000)* - JPEG-compresses the image data inside the DICOM object. If the *lossless* parameter is set to *true*, then either the improved losseless compression described above may be used or *\_standard* lossless JPEG compression may be used (DICOM PS3.5, Section 8.2.1), and the *ratio* parameter is ignored. If the *lossless* parameter is set to *false*, then lossy JPEG compression is used (also DICOM PS3.5, Section 8.2.1) as described above with the novel *CompressRT()* method. The *ratio* parameter will define the required compression ratio, which should be greater than 1.0. Moreover, the third parameter, *maxtime*, can also be used from the described *CompressRT()* implementation if

limiting compression time of a specific duration is desired. In such a case, the compression ratio would be the ratio achieved at the expiration of *maxtime*. The *Compress(bool lossless, double ratio, double maxtime)* function returns a positive integer number, which is the new size of the DICOM Object after its image part is compressed as specified. This new size is approximately equal to the original data size divided by *ratio*. The method returns -1 if JPEG compression fails for any reason.

## 2) ApplicationEntity Class

This class implements a DICOM "application entity", as described in DICOM PS3.7. An application entity is any device within the network (e.g. PC, printer, disk storage etc.) which can be involved in a DICOM image exchange. An instance of the *ApplicationEntity* class is primarily defined by its internet protocol (IP) address and port (socket) number, at which the DICOM networking occurs; this data is stored in the class' private variables. The *ApplicationEntity* class must also implement the following functions:

*bool CanAcceptCompressed()* - This function returns *true* if this application entity can accept JPEG-compressed images. This is equivalent to supporting the "1.2.840.10008.1.2.4.50" - "1.2.840.10008.1.2.4.66" and "1.2.840.10008.1.2.4.70" DICOM unique identifiers as set out in part PS3.6, Annex A of the DICOM standard. If *CanAcceptCompressed()* returns *false*, only uncompressed data can be sent to this application entity.

*bool CanAcceptLossy()* - Provided that JPEG compression is permitted (i.e. *CanAcceptCompressed()* returns *true*), this function verifies whether lossy JPEG compression is permitted at this application entity. The return value of *true* means that images with lossy JPEG compression may be accepted, which is equivalent to supporting the "1.2.840.10008.1.2.4.50" - "1.2.840.10008.1.2.4.56" and "1.2.840.10008.1.2.4.59" -

“1.2.840.10008.1.2.4.64” DICOM unique identifiers (part PS3.6, Annex A of the DICOM standard). The return value of *false* indicates that that only the images with lossless JPEG compression are accepted, which is equivalent to supporting the unique identifiers “1.2.840.10008.1.2.4.57”, “1.2.840.10008.1.2.4.58”, “1.2.840.10008.1.2.4.65”,  
5 “1.2.840.10008.1.2.4.66” and “1.2.840.10008.1.2.4.70”.

*void SetCurrentBandwidth(double bwidth)* - This function writes the bandwidth value, *bwidth*, to the corresponding private member of the *ApplicationEntity* class (with units of bytes/second). This function has no influence on the actual network bandwidth; rather, it simply records that the bandwidth was found to be equal to *bwidth*, and stores this value into a  
10 private variable of the *ApplicationEntity* class.

*double GetCurrentBandwidth()* - This function returns the network bandwidth *bwidth*, associated with the application entity, and which was previously recorded with the *SetCurrentBandwidth()* function. If no bandwidth value was previously recorded, the function returns -1.

15 *void SetMaximumCompressionRatio(bool lossless, double ratio)* - This function records the compression type and maximum compression ratio that is acceptable at the application entity. The *lossless=true* type indicates that only images compressed with a lossless algorithm can be sent to this application entity, and in this case the *ratio* parameter is ignored. The *lossless=false* type indicates that the application entity can accept lossy image compression,  
20 with the maximum compression ratio being equal to *ratio*. Whether the application entity accepts compression and to what *ratio* compression is accepted, may depend on the use of the application entity. For example, if the application entity is a computer and monitor from which physicians will view the image for diagnosis purposes, it will not be acceptable to lose



significant amounts of information in the transmitted image. Therefore, no compression may be allowed or only a limited degree of compression allowed.

*double GetMaximumCompressionRatio()* - This function returns the maximum image compression ratio, *ratio*, accepted at the application entity and which was previously recorded with the *SetMaximumCompressionRatio()* function.

*void SetDownloadTime(int time)* - This function sets the maximum download time (in seconds) in which the application entity is required to receive any DICOM object over the network. This is a download time constraint which will be set for each application entity by an operator (e.g. the network administrator) and will act as a "real-time" network requirement imposed on these particular application entities.

*int GetDownloadTime()* - This function returns the maximum object download time previously set *SetRequiredTime()*.

An alternative way to control real-time networking would be specifying download time per information unit size (i.e., *sec/Kbytes*), which is equivalent to specifying desired network bandwidth. The choice between time and time per size will depend on specific application needs.

### 3) ApplicationEntityArray Class

This class may be defined as an array of *ApplicationEntity* instances: *ApplicationEntity[MAX\_AENUM]*, where *MAX\_AENUM* is a constant specifying the maximum number of application entities possible on the given DICOM network (i.e., the total number of PCs, printers etc. at the current installation site). It also may implement the following methods:

*void SetCurrentEntity(int index)* – This function sets the current remote network application entity to entity number *index*. From the moment this function is called, all images and data will be sent from the local archive to the particular entity identified by *index*. This will continue to be the case until *SetCurrentEntity* is called again for a different application entity.

*ApplicationEntity GetCurrentEntity()* – returns the current remote application entity to which data is presently sent over the DICOM network.

#### 4) PDU Class

This class implements the DICOM Protocol Data Unit (PDU), as described in the PS3.8 part of the DICOM standard. The PDU class establishes the network connection parameters (e.g. packet headers, portion of network band over which image is sent, and data modality - e.g. CT, MRI, Ultra Sound) between the server and the application entity receiving the object or image being transmitted. As a part of this standard DICOM implementation, the class must include the following function:

*bool Send(DICOMObject& d, ApplicationEntity& ae)* – sends DICOM object *d* to application entity *ae* over DICOM network, as specified in DICOM PS3.5-PS3.8. This function returns *true* if the object successfully sent and *false* otherwise.

#### 5) NetworkTimer Class

The *NetworkTimer* class is responsible for timing current network performance. It implements the following functions:

*void StartTimer()* - starts network timer for the current DICOM network connection.

*double EndTimer()* - stops network timer for the current DICOM network connection.

These two functions, as well as the timer itself, can be implemented based on C time-processing functions such as *localtime()* and *ctime()*, which are well known in the art. Because network bandwidths will vary over time due to use conditions and other factors, it is necessary to periodically update the current bandwidth measurement. To compute the current network bandwidth, the following functions are used:

*int GetDataSize(DICOMObject& d)* - This function takes a reference to any DICOM object *d* (i.e. an image) to be sent over the network, and computes the object's size in bytes. This size is computed with *d.GetSize()* function previously defined as part of the DICOMObject class.

*void EstimateCurrentBandwidth(ApplicationEntity& ae)* - This function updates the current bandwidth value for the application entity *ae* based on observed network performance. The current bandwidth *bwidth* is computed by dividing the size, provided by the last call of *GetDataSize()* function, by time elapsed from the last call of the *StartTimer()* function (the time elapsed can be found inside the *EstimateCurrentBandwidth()* function by calling the *EndTimer()* function). Once *bwidth* is computed, this value is passed to the application entity variable *ae* with a call to the function *ae.SetCurrentBandwidth(bwidth)*.

Once the above classes and functions are established, it is possible to implement a program which will accept and maintain a maximum allowable time for downloading an image over a network. The program will monitor the bandwidth of all application entities in that network; and then when an application entity requests an image, to compress the image to the degree necessary for the application entity to download the image in less than the maximum allowable time. Such a program may be illustrated with the following segment of pseudo code seen in Figure 4-6 and written in C++ syntax. It will be understood that language following the

"/" notation are comments. The current network bandwidth for all application entities may be monitored with the pseudo code seen in Figure 4.

5 It can be seen in this code how *ApplicationEntity* *ae=ael.GetCurrentEntity()* identifies to which application entity an image is to be transmitted over the network. A PDU connection is then created between the server and the application entity. Next, from the *NetworkTimer*, the function *GetDataSize(d)* instructs the timer to prepare timing the network transmission of *d.GetSize()* bytes. Then the *StartTimer()* function will initiate the clock sequence. The *PDU* class function *Send(d,ae)* will transmit object *d* to the application entity *ae*. When the object has been downloaded by the application entity, the *EstimateCurrentBandwidth()* function will stop the clock sequence (with *EndTimer()* call). It will be readily apparent that the function *EstimateCurrentBand(ae)* may calculate the current bandwidth in bytes/sec. by dividing the size of the image by the time elapsed during transmission of the image. This measure of bandwidth will be made each time an object is transmitted to an application entity, thereby allowing a recent bandwidth measurement to be used when determining the compression ratio of the next object to be transmitted to that application entity. The pseudo code segment of Figure 5 illustrates this procedure.

20 The code seen in Figure 5 will determine if compression is necessary and allowable and if so, to what degree compression should be carried out. The function *GetSize()* determines the size of the object *d* and assigns the size value to the variable *dsize*. The maximum allowable download time for the application entity *ae* is retrieved with *GetDownloadTime()* and assigned to the variable *ntime*. A variable representing the time needed to download the object *d* with the current bandwidth, *ctime*, is assigned the value obtained by dividing the object size *dsize* by the current bandwidth for application *ae* (returned by *ae.GetCurrentBandwidth()*). If the

condition *if(ctime>ntime)* is false, then the object is transmitted with no compression. If the condition is true, the program first determines whether the application entity *ae* accepts lossy compression (*ae.CanAcceptLossy()*). If this condition is true, the lossy compression *ratio* will be determined by selecting the lesser (using the standard C function *min()*) of *ctime* divided by *ntime* or the maximum allowable compression ratio for that application entity (*ae.GetMaximumCompressionRatio()*). The object will be compressed to the degree of *ratio*. If the application entity does not accept lossy compression but does accept lossless compression, lossless compression will be applied to the object. If the application entity does not accept any type of compression, the object will be transmitted over the network in an uncompressed format, even if the object may not be downloaded within the desired time constraints.

From the above disclosure, it will be readily apparent that this embodiment of the present invention provides a method of adjusting the compression ratio of image data based upon the available bandwidth of a network across which the image is transmitted

The invention also encompasses an alternative method for implementing the time functions *StartTimer()* and *EndTimer()*. Above, the standard C *localtime()* and *ctime()* functions were utilized, but these functions only measure time as an integer (i.e. as a whole second). An alternative implementation, which would measure time within fractions of a second, is possible with Windows API. The class is designated *AccurateTimer* and is illustrated with the pseudo code seen in Figure 6. However, it is noted that the *AccurateTimer* class is known in the prior art and is shown here as an example, and not as part of the present invention.

**Sound Encoding.** The present invention further includes a method of encapsulating audio data in a DICOM object such as an image. This is accomplished by creating a SoundRecorder object which performs sound recording and playback. The sound is captured and played by the sound recording hardware, including PC sound card, microphone and speakers. When the sound is recorded, it is sampled to the optimal digital frequency chosen by the recording clinician. The following methods must be implemented:

```
bool Record(int sf, bool stereo, int max_time=300, int max_size=1000000, int format=WAV)
```

This function records sound input with the following specified digital sampling parameters:

*sf* - Sampling Frequency, KHz  
*stereo* - Stereo (yes or no)  
*max\_time* - Maximum recording time in seconds  
*max\_size* - Maximum record size in bytes  
*format* - Digital sound format

This sound-recording function may be implemented based on many conventional operating system sound interface supports well known in the art. For instance, Windows implementation for basic sound recording would appear as seen in Figure 7. Playing back the recorded sounds may readily be accomplished by a function such as:

*bool Playback(int from, int to)* – play back recorded sound, from a first time *from* to second time *to*. In other words, the function parameters are:

*from* – Playback start position  
*to* – Playback end position

For instance, on a Windows platform, a simple playback implementation may be accomplished using the standard Windows *PlaySound()* function.

SoundRecorder Object.

After the sound has been recorded, it may be advantageous to convert the sound into a different format (for instance, from WAV to MP3 format). For example, conversion from WAV to MP3 format may reduce the size of sound data by a factor of approximately 10 without any perceivable loss of quality. This conversion may be made (after the sound is recorded) with an appropriate codec (e.g., an MP3 encoder), incorporated into the below described *SoundRecorder* object.

This object encodes/decodes digitized sound in the well-known MP3 format. For encoding process, the original digitized sound is received as *SoundRecorder* object output after recording. For decoding process, the sound is converted back from MP3 to the digital format accepted by *SoundRecorder* (for instance, WAV format on Windows platform), and is played back. Therefore, the following functionality must be implemented:

*BYTE\* GetSound()* – returns the pointer to the current sound (stored as a BYTE buffer).

*int GetFormat()* – returns the current sound format (WAV, MP3, etc.)

*bool Insert(DICOMObject& dob)* – insert this sound into DICOM Data Set. The return value of *true* confirms successful insertion, and *false* corresponds to failed insertion. The implementer will have to specify a 4-byte group/element tag number (PS3.5 part of DICOM standard) to identify the sound entry in his DICOM dictionary – since there is no sound entry in the standard DICOM dictionary.

*bool Encode(BYTE\* input, int inputFormat, BYTE\* output)* – this function takes a byte stream *input*, which represents a digitized sound in *inputFormat* format (such as WAV), and encodes it into MP3 stream *output*. The function returns *true* if encoding process was successful, and *false* otherwise.

*bool Decode(BYTE\* input, int outputFormat, BYTE\* output)* – opposite to *Encode*, decodes MP3 stream *input* into output stream *output* with *outputFormat* format. The *SoundRecorder::Insert()* function is implemented as seen in Figure 8.

5 It may also be desirable at some point in time to remove a previously inserted sound from an object. This removal may be accomplished with the following function:

*bool Extract(DICOMObject& dob, bool remove)* – extract digitized sound buffer from DICOM Data Set *dob*. If the *remove* parameter is set to *true*, the sound is removed from *dob* after it is extracted; otherwise, an original copy of the sound buffer will remain in *dob*.

10 Thus, with the above described code, audio data may be encapsulated into a DICOM object such as a patient study, the data converted to more compact formats, and if desired, the audio data may also be removed.

### **Integer-Based Ray Tracing**

15 Ray tracing or ray casting is a well known technique for using three-dimensional (3-D) image data to draw a 3-D appearing object on a two dimensional (2-D) imaging device (e.g. the screen of a computer monitor). To give the object displayed on the 2-D screen the appearance of depth, different areas of the object must be shaded. Ray tracing uses information from the 3-D data to determine what areas of the object on a 2-D screen should be shaded and to what degree the areas should be shaded. Such prior art methods of Ray Tracing are set out in *Radiosity and Realistic Image Synthesis*, Cohen and Wallis, Academic Press, Inc., 1993, which  
20 is incorporated by reference herein.

One typical example of a 3-D data set could be a series of CT scan images as suggested in Figure 9. Each of the three images in Figure 9 represents a "slice" taken at a certain level of a human head. A complete series of CT scan images for a human head may comprise 100 to



200 such slices. It will be understood that each slice provides information in two dimensions, for example the x-y plane. By supplying multiple slices, the series of CT scans in effect provide information in the third dimension or the z direction. Different tissues on the CT image are represented by different color and/or intensity (i.e. brightness). For every position (x, y, z) in the 3-D image data, there is a corresponding intensity value ( $L$ ). Thus, the 3-D image data may be represented by the equation  $F(x, y, z) = L(x, y, z)$ . When it is desired to view a certain tissue type or object of interest (e.g. a suspected tumor) corresponding to a constant intensity of  $L$  in the CT image, this may be accomplished by identifying all (x, y, z) points in the 3-D data where  $F(x, y, z) = L$ .

Figure 10 schematically illustrates a 3-D data set 10 having an object of interest 12 ("object 12") located within 3-D data set 10. A 2-D image plane 14 is shown with several lines of sight or "rays" 18 extending between 2-D image plane 14 and observer position 16, while also passing through 3-D data set 10. While not shown, it is presumed for the purposes of this explanation that the light source is directly behind observer position 16. A ray 18 will be back projected from the picture elements or pixels on 2-D image plane 14 to observer position 16. While this process of "ray generation" could be performed for each pixel on 2-D image plane 14, it is generally sufficient to generate rays from every nth pixel (e.g. every third or fourth pixel) in order to save computational time. Where a ray 18 (viewed from observer position 16) intersects object 12, it is known that the corresponding pixel on the 2-D image plane 14 will have an intensity value related to the brightness or shading of object 12 as seen from observer position 16. Pixels associated with those rays 18 not intersecting object 12 will have predetermined background intensity, such as zero representing a completely black pixel.

To determine the proper shading of those pixels on 2-D image plane 14 associated with object 12, it must be determined what the angle of the surface of object 12 is relative to the ray 18 impacting object 12 at that point. As noted above, the light source is presumed to be directly behind observer position 16. Thus, if the surface of object 12 at the point of ray impact is normal to that ray (and thus observer position 16), then the maximum light will be reflected back from that point to observer position 16. The pixel on 2-D image plane 14 corresponding to that ray 18 will consequently be assigned the highest intensity value. On the other hand, if the surface at a point of impact of a ray 18 is parallel to that ray, then virtually no light will be reflected and the pixel associated with that ray 18 will have very low intensity value.

To determine the surface angle at a point of impact, it is necessary to determine the distance " $t$ " from the surface of the object to observer position 16 (or another point along the ray 18) and compare that to the same distance for adjacent points of impact. Figure 11 illustrates an enlarged section of object 12 with several rays 18 impacting its surface. The length  $t$  for each ray is shown as taken from an arbitrary point W along the rays 18. It is irrelevant whether point W is observer position 16 or a point much closer to object 12. It is only necessary that point W be displaced from the surface of object 12 such that it is possible to measure the relative distances between point W and the point of impact for neighboring ray 18. This method of determining where along ray 18 object 12 is encountered (i.e. the length of  $t$ ) is a form of data volume traversal and is described further below.

Once  $t$  is determined for a ray 18 and several neighboring rays, it is possible to determine the surface angle at ray 18's point of impact. Figure 12 is a graphical representation illustrating how the surface angle of a point of impact T will be calculated using the relative distance between observer position and points of impact T, A, B, C, and D. Points A, B, C, and

D are points of impact neighboring point T. Points A, B, C, and D have relative distances to the observer position of  $a$ ,  $b$ ,  $c$ , and  $d$  respectively. T1, A1, B1, C1, and D1 are points at distances  $t$ ,  $a$ ,  $b$ ,  $c$ , and  $d$  respectively above the surface of object 12. It can be seen that points T1, A1, and D1 form a triangular plane (shown in dashed lines) and this triangular plane has a vector 20 which is normal to the surface of the triangular plane. While not shown in Figure 12 in order to maintain simplicity, it will be understood that similar triangular planes and normal vectors 20 will be computed for triangles T1 A1 B1, T1 B1 C1, and T1 C1 D1. The normal at point T is then approximated as the average normal of the four normal vectors 20. The estimated normal vector at T will be the vector  $\{(a-c)/2, (d-b)/2, 1\}$ . It is this normal vector at point T, which is considered the normal to the surface angle at the point of impact.

As alluded to above, the intensity of a pixel on the 2-D image plane is determined by that pixel's associate ray 18 and the surface angle on the object relative to that ray 18. In other words, the intensity of a pixel will be related to the angle between the associated ray 18 and the normal vector at the point of impact. If the ray 18 and normal vector are parallel, the surface directly faces the observer point (and light source) and the point of impact reflects maximum light (i.e. the pixel is assigned a high intensity value). If the ray 18 and normal vector or perpendicular, the point of impact reflects very little or no light and the associated pixel is assigned a low intensity value. When the angle ( $\alpha$ ) between ray 18 and the normal vector is between  $0^\circ$  and  $90^\circ$ , the intensity is determined as the maximum intensity multiplied by cosine  $\alpha$ .

The two main components of the above described prior art method of ray tracing (ray generation and volume traversal) are computationally very demanding and often require

excessive amounts of computer processing time. With ray generation, each ray must be defined by a set of equations such as:

$$\begin{aligned}xray &= a + t * l1 \\ yray &= b + t * l2 \\ zray &= c + t * l3\end{aligned}$$

where *xray*, *yray*, *zray* are the components of a ray 18 (such as to the point of impact), *a*, *b*, *c* are the coordinates of the observer position, *t* is the relative distance (i.e. *xray*, *yray*, *zray* to the point of impact), and *l1*, *l2*, *l3* are values representing the direction of ray 18. Generally, these numbers will be large floating point values and considering a ray is generated for large number of pixels, the size of the numbers will greatly add to the processing time required to ray trace an image.

When finding the distance "*t*" from the surface of object 12 to observer position 16 as described above, prior art ray tracing techniques typically use a form of volume traversal. As illustrated conceptually in Figure 13(a), a 3-D data set 10 is provided with a ray 18 beginning at observer point (plane) 16 and extending through data set 10. Data set 10 may be viewed as being sub-divided into a series of cells 26. Figure 13(b) illustrates how each cell 26 will be formed from a 8 points T1-T8. It will be understood that each point T1-T8 is a known data point. For example, referring back to Figure 9, T1-T4 would be adjacent pixels on one CT image or slice and T5-T8 would be the same pixels on an adjacent slice. In the volume traversal method, each cell 26 which is intersected by ray 18 must be analyzed. It must first be determined whether a cell 26 contains any portion of the object of interest. If so, and if further the ray 18 actually intersects the object within the cell, then it must be determined exactly where along the length of ray 18 (within cell 26) that the object is intersected. This point of intersection provides the length "*t*". Of course, if the cell 26 does not contain part of object 12,

then the next cell along ray 18 is analyzed and this process is continued for all cells 26 (along ray 18) until object 12 is encountered or ray 18 exits 3-D data set 10. It will be apparent that carrying out this data volume traversal for many thousands of rays is a computationally expensive procedure, particularly when carried out with floating point values as practiced in the prior art.

The prior art has made attempts to reduce the processing time expended on ray generation and data volume traversal. One alternative developed is to project rays from every  $n$ th pixel (say every 3rd, 4th or 5th pixel) rather than every pixel. Once a determination of the intensity of every  $n$ th pixel is made, an interpolation may be made for the values between every  $n$ th pixel. Naturally, the larger the value of  $n$  is, the more error possibly introduced into the final image. Additionally, the various methods of interpolation have their own disadvantages. Two manners of interpolation known in the art are Gouraud's linear interpolation and complex Fong interpolation. However, Gouraud interpolation produces artifacts on the square boundaries. While complex Fong interpolation provides more accurate results, its complexity expends much of the computation savings gained by choosing every  $n$ th pixel. Moreover, even when generating rays only for every  $n$ th pixel, it is still necessary to generate many thousands of rays to accurately portray an image.

The present invention provides a much more computationally efficient method of ray tracing than hereto known in the art. This method effectively replaces the large floating point numbers with integers, allowing all calculations to be integer based and therefore much faster. Figure 14 illustrates another example of a 3-D data set 10 having an object of interest 12 and a 2-D image plane 14 conceptually positioned behind 3-D data set 10. However, Figure 13 differs from the previous example illustrated by Figure 10 in that there are not multiple rays

from observer position 16 to each pixel on 2-D image plane 14. Rather, there is a single observer ray 23 which extends from image plane 14 to observer position 16. As in the example above, observer ray 23 may be defined by the equations:

$$\begin{aligned}xray &= a + t*l1 \\ yray &= b + t*l2 \\ zray &= c + t*l3.\end{aligned}$$

Now however, for purposes explained below,  $xray$  will be set equal to  $i$ , resulting in the set of equations:

$$\begin{aligned}xray &= i \\ yray &= b + (i-a)*l2/l1 \\ zray &= c + (i-a)*l3/l1\end{aligned}\quad \text{Equation (4)}$$

Next, the values  $a$ ,  $b$ ,  $c$ , together with the ratios  $l2/l1$  and  $l3/l1$  will be converted to rational numbers (i.e. numbers which are a ratio of two integers) which closely approximate the original floating point values. By changing the values  $a$ ,  $b$ ,  $c$ , together with the ratios  $l2/l1$  and  $l3/l1$  to rational numbers, the computational expense of operating upon floating point numbers may be eliminated. While this will slightly change the observer position and ray angle represented by the values  $l1$ ,  $l2$ ,  $l3$ , the changes are far too small to adversely affect the final results of the ray tracing. For example, a floating point value of  $a=2.3759507...$  could be represented by the rational number 2.37 (i.e. the ratio of the integers 237 and 100). Because the substituted rational numbers are such close approximations to the original floating point values, the change in observer position cannot be noticed by a user and the ray angle still allows the rays to impact the same pixel on the 2-D image plane.

As additional rays are back projected from image plane 14, the method of the present invention does not converge these rays on observer point 16. Rather these rays 24 travel parallel to observer ray 23 (i.e., have the same values for  $l1$ ,  $l2$ , and  $l3$ ). These parallel rays 24

will extend to origin points 22 in the same plane in which observer point 16 lies and which is perpendicular to rays 24. Like observer point 16, each origin point 22 will have its  $a$ ,  $b$ ,  $c$  coordinates adjusted to integer values. This process is repeated for all other pixels on image plane 14 from which rays 24 will be back propagated.

5        The main practical outcome of this method of substituting rational values for floating point values is that it allows using only integer additions for ray generation and data volume traversal. This process is described by equation (4), where fractions like  $l2/l1$ ,  $l3/l1$  or  $l2/l3$  were assumed to be rational. This is identical to considering  $l1$ ,  $l2$  and  $l3$  as integers, since in the case of using rational numbers, both sides of the equation (4) can be multiplied by the least  
10   common denominator of  $l1$ ,  $l2$  and  $l3$ , which will eliminate any fractional parts. Note that numbers  $l1$ ,  $l2$  and  $l3$  are constants, and so is their common denominator, therefore the conversion from rational to integer is carried out only once, and does not need to be recalculated for each particular ray.

Figure 15(a) shows with a 3-D data set 10 and an individual cell 26 within data set 10.  
15   As with Figure 14(b), Figure 15(b) shows an enlarged view of cell 26 with corners T1-T8 representing known data points. It will be understood that the component of a ray 24 along the x-axis (i.e. the  $xray$  value from Equation (4)) will be set equal to  $xray=i$ . Equation (4) is written for the case where  $xray=i$  results in the ray intersecting the side of cell 26 closest to origin 22. In Figure 15(b), point A illustrates the point where ray 24 intersects this side of cell  
20   26 upon entry of cell 26 and point B illustrates the cell side where ray 24 exits cell 26. It will be understood that because equation (4) is composed of rational numbers, the position of points A and B must also be rational numbers with the same denominator, which also permits consideration of all these numbers as integers.

The fact that rays produced by equation (4) are traced through to a point located on a cell side has its own benefit of minimizing the complexity of linear interpolation. If, for instance, the point A was not located on a cell side, one would have to use all 8 cube vertices T1-T8 for linear volume interpolation of  $F(A)$ . But when A is a side point we use only two-dimensional linear interpolation of  $F(A)$  involving 4 points T1-T4 that cuts calculation cost and time by more than twice. Because the function  $F(x,y,z)$  was chosen as integer on the integer grid points  $(x,y,z)$ , its value is rational at a rational point A and has therefore the form of  $F(A)=F1(A)/F2(A)$ , where integer numbers  $F1(A)$  and  $F2(A)$  can be found with integer arithmetic directly from the linear interpolation formula.

The location of each point A on a cell is defined by three distances AA1, AA2 and AA3, measured from A to the closest cell side in the y, z and x dimension respectively. Because A resides on a cell side, at least one of these coordinates (e.g., AA3 on the opposite cell wall) is equal to the cell size in this dimension. The size of the cell in all dimensions is an integer value. The remaining coordinates (AA1 and AA2) are integers as well. When the ray point A needs to be traced to the next position B, it's equivalent to solving Equation (4) for another integer point  $(B1,B2,B3)=(xray,yray,zray)$ , which also implies that  $(B1,B2,B3)$  is obtained as an integer offset of  $(A1,A2,A3)$ . Thus, ray tracing becomes a purely integer based calculation, and volume traversal is reduced to simple and time-efficient integer subtraction.

The next step is the computation of ray intersections with the surface  $F(x,y,z)=L$  (the intensity value of the object of interest). For this equation and linear interpolation properties, a cell will contain an intersection point if and only if some of  $F(T1)$ , ...,  $F(T8)$  have different values greater than or less than  $L$ . In other words, if  $F(T1)$ , ...,  $F(T8)$  are all values either



greater than  $L$  or less than  $L$ , it will be presumed that the cell does not contain a point having a value of  $L$ . This simple criterion is used to efficiently reject the cells with no intersections.

Once some of  $F(T1)$ , ...,  $F(T8)$  have different values greater and less than  $L$ , an intersection point has to be inside the cell, and may be on the  $[AB]$  segment. In this case, the values of  $F(A)$  and  $F(B)$  are also computed, and compared to  $L$ . If  $F(A)$  and  $F(B)$  are both smaller or both larger than  $L$ , the segment  $[AB]$  contains no intersection point, and the tracing continues to the next cell. Otherwise an intersection point belongs to  $[AB]$ , and its distance from the ray origin must be computed with a proper interpolation. It will be understood that a 3-dimensional linear interpolation, used to compute  $F(x,y,z)$  at any volumetric point, taken along a ray, becomes one-dimensional cubic interpolation. Thus, the prior art method of solving for the root of  $F(x,y,z)=L$  requires the solution of a third-order equation. A typical approach taken in the prior art to reduce the complexity of this solution was to replace the cubic equation by its linear approximation. However, this approach creates visible surface distortions. The present invention utilizes a simple second-order approximation, and not to the original third-order equation, but to the formula for its root (inverse interpolation) as suggested in Figure 16. To perform this approximation, the method must:

- Compute  $F(M)$ , where  $M$  is the midpoint of the segment  $[AB]$ ;
- Consider three points  $(tA, F(A))$ ,  $(tM, F(M))$  and  $(tB, F(B))$  and pass a parabola through them. Since all values are integers, the parabola will have rational coefficients (where  $tA$ ,  $tM$ , and  $tB$  are distances to points  $A$ ,  $M$ , and  $B$  respectively); and
- Take the lower-order (constant) term of this parabola, as the approximation to the root  $t0$ .

where  $t_0$  is the value which will represent the distance to the object surface (i.e. distance " $t$ " discussed above. This yields the following rational root approximation formula:

$$t_0 = \frac{F(M)(tA - tB) + F(A)(tB - tM) + F(B)(tA - tM)}{F(M)F(B)(F(M) - F(B)) + F(A)F(B)(F(B) - F(A)) + F(M)F(A)(F(A) - F(M))}$$

This interpolation proved to be more accurate than the linear one, but still requires only integer computations. The parabolic interpolation was found to take only 7-10% more time compared to the linear interpolation, but produces much less distortion.

As discussed above, rays need not be traced for each point, but rather may be taken at some predefined step  $d$  in each dimension, where  $d$  is optimally chosen (for example, only each  $d$ -th ray is traced in each dimension). However, rather than employing the prior art Gouraud's linear interpolation or complex Fong interpolation, the present invention uses a simple but sufficient bi-quadratic interpolation for shading or pixel intensity. Figure 17 represents a grid of pixels where  $d = 4$  and A, B, C, D, and p1-p8 are intensity values from rays traced at every fourth pixel. The purpose of the pixel intensity interpolation is to compute the intensity for all pixels (in this case nine) inside the  $d \times d$  ABCD square, which correspond to the "skipped" rays. As mentioned above, Linear Gouraud's interpolation performs this computation based only on four intensities at points A, B, C and D, thus producing well-known artifacts on the square boundaries. The present invention uses a second order 2-D approximation polynomial:

$$a_{20}x^2 + a_{02}y^2 + a_{11}xy + a_{10}x + a_{01}y + a_{00},$$

where the coefficients are chosen by a conventional least squares regression technique which gives the exact intensities at points A-D, and still produce the closest least-squares match to the intensity values at p1-p8. This method gives a smoother interpolation on the square boundary with marginal processing time overhead.

As  $d$  is increased, tracing becomes faster (since only each  $d$ -th ray must be fired in each dimension), but parabolic intensity interpolation becomes generally slower. Experimentation was carried out with different  $d$  values from 2 to 15, observing the shading times for different test objects. The results are shown in Figure 18. It can be seen that the most substantial time reduction occurs before  $d=4$ . Increasing  $d$  further not only starts to consume more execution time, but also produces visible image artifacts. Therefore, a step size of  $d=4$  is believed to be most efficient both in terms of quality and speed of ray tracing. Submitted with this application is a source code listing which implements the novel ray tracing method described above.

#### **Advanced Magnifying Glass Tool**

Because radiological images must be so carefully scrutinized, it is common for prior art software used in viewing such images to provide a magnifying window. Figure 19 illustrates a viewing screen 30 (such as a high resolution computer monitor) having a medical image 31 displayed thereon. A magnifying window 32 is shown displaying an enlarged portion of image 31. When in use, magnifying window 32 is typically moved with a "mouse" or similar device by conventional "click and drag" techniques. When window 32 is moved to a new position, the center of the window marks the center of original image region to be magnified. Based on the power of the magnification and the window size, the magnifying window program computes the area of the original image to be magnified. For example, if the magnifying window is 3"X 3", and the power of magnification is 2x, the region on the original image to be magnified will be 1.5 " x 1.5 ".

When the user moves the window to a new position, the prior art magnifying software updates the window position by performing a series of steps as illustrated in Figures 20(a)-20(c). Figure 20(a) shows an initial position of window 32 designated as  $P_0$ . When the user

begins to drag the window to a new position, the program first removes the initial window  $P_0$ . Next, the program restores to the screen the original image region covered by window  $P_0$  (dashed area 33 in Figure 20(b)). After calculating the next region of the original image covered by the new window position  $P_1$ , the program magnifies that region and displays it within the window position  $P_1$ . Although Figures 20(a)-20(c) can only illustrate this updating process as two distinct window positions, it will be understood that the updating process occurs continuously as the user moves the magnifying window 32 across the viewing screen. The actual speed at which the individual updates occur will depend on factors such as the processing speed of the computer and the complexity of the image. If the updates do not occur with sufficient rapidity, the user will be able to detect discontinuities in successive window positions as the window moves across the screen. This phenomenon is commonly referred to as "flicker." Flicker is not only aesthetically undesirable, but for health care providers who must view medical images during an entire work day, flicker can significantly contribute to viewer distraction and fatigue. Additionally, magnifying an image may tend to blur or distort the image. Prior art magnifying programs typically fail to optimize the contrast of the enlarged portion of the image and fail to filter distortions caused by the magnification. A magnifying window program which reduces flicker and reduces distortions caused by magnification would be a significant improvement in the art.

Figure 21 illustrates an improved update method for a magnifying window and Figure 22 shows a flow chart for carrying out this method. As a simplified example, Figure 21 represents a 500 x 500 pixel screen 30 with an initial window position  $P_0$  and an updated window position  $P_1$  moved 50 pixels upwards and to the left. It will be understood that in actual operation, the window  $P_1$  would be updated after moving only a small distance equal to a

few pixel positions, since mouse positions are sampled very frequently. However, to make the illustration clearer, Figure 21 is shown with an exaggerated change in distance between windows  $P_0$  and  $P_1$ . Figure 22 illustrates the first step 35 in the method is to store the original image in memory. When the magnifying window routine is called, the program will first  
5 determine the region ( $R_0$ ) of the original image to be magnified (step 36) and then display the region  $R_0$  (as magnified) in initial window position  $P_0$  (step 37). When the user next moves the magnifying window to a new position, the program will receive the new window position  $P_1$  and then calculate new region  $R_1$  to be magnified (steps 38 and 39). Next the program determines in step 40 which corner of  $P_1$  is found in  $P_0$ . As suggested in Figure 21, the corner  
10 located at the pixel position (100,100) is the corner of  $P_1$  found in  $P_0$ . With this information and the pixel positions of the  $P_0$  corners, the program in step 41 is able to define two rectangular regions A and B which are the portions of  $P_0$  no longer covered by  $P_1$ . The program then retrieves from memory the original image information for the regions A and B and only needs to restore the original image to these areas rather than to the entire area of  $P_0$ .  
15 With frequent magnifying window updates, the areas of rectangles A and B are much smaller compared to that of  $P_0$ . Therefore, this method of restoring only the regions A and B greatly reduces the likelihood that any type of flicker will be perceived by the magnifying window user.

Additionally, the magnifying window of the present invention allows further processing  
20 of the image as indicated by steps 43-47 in Figure 22. When the user wishes to optimize the contrast and filter the magnified image, he or she may select this further processing option by activating the appropriate indicator which will appear on or next to the magnifying window. As indicated in steps 43-47, this processing will include optimizing the contrast by removing

outlying pixels and redistributing the pixels over the intensity range of the display system. Figures 23(a)-23(c) graphically illustrate this processing. In Figure 23(a), a histogram of the image region being magnified is produced by plotting pixel intensity versus the frequency at which pixels of that intensity occur. In these figures, the intensity scale ranges from 0 to 255.

5 Figure 23(b) represents how a given percentage of the total pixels will be removed from the extreme ends of the intensity scale. In the illustrated example, the outlying 1% of the pixels are removed, however, a lesser or greater percentage of the outlying pixels could be removed depending on the degree to which it is desired to enhance contrast. Figure 23(c) illustrates how the image region is re-scaled to distribute the remaining pixel over the entire 0 to 255 intensity

10 range. After enhancing the contrast of the image region, step 47 in Figure 22 shows how filtering of the image region will take place before the image region is displayed in window P<sub>1</sub>. While various forms of filtering may be implemented, it has been found that a conventional median filter provides favorable results. As those skilled in the art will recognize, median filters operate by examining the neighborhood intensity values of a pixel and replacing that

15 pixel with the median intensity value found in that neighborhood. The application of different filters transforms a conventional magnifying glass into a more powerful image analysis instrument. This is particularly beneficial when large images are analyzed, and processing the entire image becomes a time-consuming and unnecessary process.

Computer programs based on all of the methods and pseudo code disclosed above can

20 be run on any number of modern computers systems having a suitable computer processor and a sufficient quantity of computer-readable storage medium. Although certain preferred embodiments have been described above, it will be appreciated by those skilled in the art to which the present invention pertains that modifications, changes, and improvements may be

made without departing from the spirit of the invention as defined by the claims. All such modifications, changes, and improvements are intended to come within the scope of the present invention.

I Claim:

1. A computer-readable storage medium containing computer executable code for instructing a computer to perform a method of lossless image compression comprising the steps of:

- a. taking a sample of pixel neighborhoods from an image file;
- b. using said sample of pixel neighborhoods to determine a series of predictive coefficients values related to said pixel neighborhoods;
- c. determining prediction residual values based on said coefficients values; and
- d. losslessly compressing said prediction residual values.

2. The computer-readable storage medium according to claim 1, wherein coefficient values  $c_{i,j}$  and said prediction residual values  $e(x,y)$  are determined from the equation:

$$I(x,y) = \sum_{0 \leq i,j \leq a} c_{i,j} I(x-i, y-j) + e(x,y),$$

3. A computer-readable storage medium containing computer executable code for instructing a computer to perform a method of adjusting the quality parameter in a quality controlled compression routine in order to compress image data a predetermined compression ratio, said method comprising the steps of:

- a. receiving a desired compression ratio;
- b. selecting an estimated quality value;
- c. compressing said image data based on said quality value and calculating an intermediate compression ratio from said compressed image data;
- d. adjusting said quality value in iterative steps and recalculating said intermediate compression ratio; and
- e. returning a final quality value after a predetermined time period or when said predefined ratio is achieved;



f. compressing said image data to said final quality value using said quality controlled compression routine.

4. The computer-readable storage medium according to claim 3, wherein said estimated quality value is selected from a predetermined maximum quality and a predetermined minimum quality.

5. The computer-readable storage medium according to claim 4, wherein said intermediate compression ratio during an iteration is the dividend of an original image size divided by the size of an image compressed according to the quality value of said iteration.

6. A computer-readable storage medium containing computer executable code for instructing a computer to perform a method of adjusting the compression ratio of image data based upon the available bandwidth of a network link across which said image is transmitted, said method comprising the steps of:

- a. determining the size of an image to be transmitted;
- b. selecting a desired download time for downing loading said image from a network link;
- c. determining a current download time based upon current network conditions;
- d. compressing said image if said current download time is greater than said desired download time.

7. The computer-readable storage medium according to claim 6, wherein said image is compress by a ratio greater than or equal to a ratio of said current download time divided by said desired download time.

8. A computer-readable storage medium containing computer executable code for instructing a computer to perform a method of encapsulating audio data in a DICOM object and selectively retrieving said data, said method comprising the steps of:

- a. providing a DICOM compliant recording function having parameters representing a recording time and a record size of said audio data;
- b. providing a DICOM compliant playback function having a parameter representing a playback start position.

9. The computer-readable storage medium according to claim 8, further comprising the step of providing a DICOM compliant function for converting audio data from a first format into a second format.

10. A computer-readable storage medium containing computer executable code for instructing a computer to perform an improved integer based ray tracing method for constructing two-dimensional images from three-dimensional data, said ray tracing method comprising the steps of:

- a. receiving a set of three-dimensional image data containing an object of interest;
- b. receiving an observer position;
- c. establishing a ray angle from said observer position, through said three-dimensional data, to a two-dimensional image plane;
- d. adjusting said ray angle such that the directional components of said ray angle are rational numbers;
- e. back projecting from a selected number of picture elements on said two dimensional image plane a series of rays parallel to said ray angle, said rays passing through said three-dimensional image data to origin points;

f. for each ray intersecting said object of interest, determining a relative distance between said origin point and a point of contact where a ray intersects said object of interest;

g. determining a surface angle relative to said origin point for each of said points of contact;  
and

5 h. adjusting the intensity of said picture elements on said two dimensional image plane relative to said surface angle of said points of contact.

11. The computer-readable storage medium of claim 10, further including the step of adjusting the origin point of each ray such that the coordinates of said origin point are rational numbers.

10 12. The computer-readable storage medium of claim 10, wherein said step of determining a relative distance between said origin point and said point of contact includes the step of making a parabolic interpolation.

13. The computer-readable storage medium of claim 10, wherein said step of adjusting the intensity of said picture elements further includes an interpolation based on a second order polynomial.

15 14. In a computer system including a computer processor, a memory, and a display screen, a method of reducing flicker caused by a magnifying window moving across an image on said display screen, said method comprising the steps of:

a. storing said image in said memory;

b. storing a first window position in said memory;

20 c. reading a second window position, which overlaps said first window position;

d. determining what portion of said first window position is not covered by said new window position;

e. restoring from memory that portion of said image which corresponds to said portion of said first window not covered by said second window.

15. The method according to claim 14, further including the step of filling said first and second window positions with a magnified portion of said image.

5 16. The method according to claim 14, wherein said step of determining what portion of said first window position is not covered by said new window position further includes the step of dividing said uncovered portion into two rectangles.

10 17. The method according to claim 14, further including the step of removing outlying pixel values from a region of said image to be magnified and redistributing remaining pixel values of said region across an intensity spectrum of said computer system.

18. The method of claim 17, further including the step of applying a median filter to said region of said image to be magnified.

19. A radiologist workstation stored on a computer-readable storage medium containing computer executable code for instructing a computer to perform the methods of:

- 15 a. losslessly compressing an image by using a sample of pixel neighborhoods to determine a series of predictive coefficients values related to said pixel neighborhoods;
- b. lossy compressing an image to a predetermined ratio by adjusting the quality parameter in a quality controlled compression routine;
- c. adjusting the compression ratio of image data based upon the available bandwidth of a
- 20 network link across which said image is transmitted;
- d. encapsulating audio data in a DICOM object and selectively retrieving said data;

- 5
- e. performing an improved integer based ray tracing method by establishing a ray angle from an observer position and adjusting said ray angle such that the directional components of said ray angle are rational numbers; and
  - f. reducing flicker caused by a magnifying window moving across an image on said display screen by determining what portion of a first window position is not covered by a new window position and restoring from memory that portion of said image which corresponds to said portion of said first window not covered by said second window.

## ABSTRACT

A radiologist workstation program capable of performing the methods of: (a) losslessly compressing an image by using a sample of pixel neighborhoods to determine a series of predictive coefficients values related to the pixel neighborhoods; (b) lossy compressing an image to a predetermined ratio by adjusting the quality parameter in a quality controlled compression routine; (c) adjusting the compression ratio of image data based upon the available bandwidth of a network link across which the image is transmitted; (d) encapsulating audio data in a DICOM object and selectively retrieving the data; (e) performing an improved integer based ray tracing method by establishing a ray angle from an observer position and adjusting the ray angle such that the directional components of the ray angle are rational numbers; and (f) reducing flicker caused by a magnifying window moving across an image on the display screen by determining what portion of a first window position is not covered by a new window position and restoring from memory that portion of the image which corresponds to the portion of the first window not covered by the second window.



- 1. Drawings**
- 2. Oath or Declaration**
- 3. Foreign language specification**
- 4. Sequence listing**
- 5. Computer listing**
- 6. Appendices**

# 2

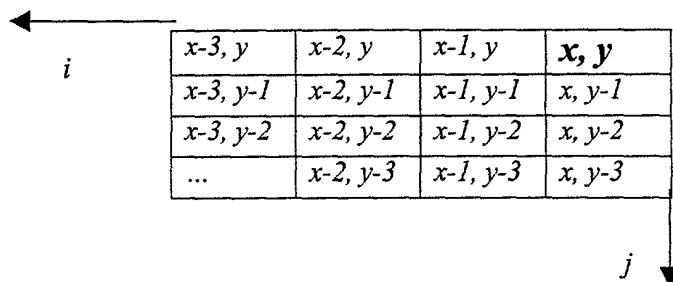


FIGURE 1

```

// for each pixel in the NxM image
for(x=1; x<N; x+=1)
{
    for(y=1; y<M; y+=1)
    {
        // Consider I(x,y), I(x-1,y), I(x,y-1), ... neighborhood of (x,y)
        // If some if I(x-1,y-j) fall out of image boundaries, replace them by 0
        // Do not consider reference pixels (I(x-i,y-j) where i,j=0)
        // Apply (6) to compute e(x,y)
        sum_kI=0;
        for(i=0; i<a; i+=1)
        {
            for (j=0; j<a; j+=1)
            {
                if(I(x-i,y-j)==out of bounds pixel) I(x-i,y-j)=0;
                if(i==0 && j==0) continue;
                sum_kI=sum_kI+kij*I(x-i,y-j);
            }
        }
        sum_kI=sum_kI>>10;
        e(x,y)=I(x,y)-sum_kI;
    }
}

```

FIGURE 2



```

int RTJPEG::CompressRT(BYTE* input, BYTE* output, double ratio, double maxtime,
int bpp=1, bool rgb=false)
{
    int size_orig = Size(input); // original image size in bytes
    // Set max and min quality values
    int q0=5; // 5% quality
    int q1=95; // 95% quality
    // Find respective compression ratios. Higher ratio corresponds to lower quality
    double r0 = size_orig/Compress(input,output,q0, int bpp=1, bool rgb=false);
    // highest ratio
    double r1 = size_orig/Compress(input,output,q1, int bpp=1, bool rgb=false);
    // lowest ratio
    // Return if proposed ratio is smaller than the smallest possible
    if(ratio<r1) return JPEG::Compress(input, output, q1);
    // Return if proposed ratio is larger than the largest possible
    if(ratio>r0) return Compress(input, output, q0);
    // Start timer
    clock_t start, finish;
    start = clock();
    // Start iterative process to estimate the quality value
    double duration=0; double r = ratio;
    while (duration<maxtime)
    {
        // Use linear quality estimate
        int q = q0+(r-r0)*(q1-q0)/(r1-r0);
        // Alternatively, a bi-section estimate can be used in the previous line:
        // int q = (q0+q1)/2;
        // Update estimated corresponding compression ratio value
        r = size_orig/Compress(input,output,q);
        // Update compression and quality boundaries
        if(ratio>r)
        {
            r1 = r; if(q1==q) break; // convergence
            q1 = q;
        }
        else
        {
            r0=r; if(q0==q) break; // convergence
            q0=q;
        }
        // Update timer
        finish = clock(); duration = (double)(finish - start) / CLOCKS_PER_SEC;
    } // end of iterative process
    // Compress
    return Compress(input, output, q);
}

```

FIGURE 3

```

//      It is assumed that ApplicationEntityList ael and DICOMObject d variables
//      have been already initialized based on the present network
//      configuration and the data to be sent.

// Find current remote application entity, to which data must be sent
ApplicationEntity ae = ael.GetCurrentEntity();

// Create PDU connection
PDU pdu;

// Start network performance timer
NetworkTimer nt;

// Find data size to be sent
nt.GetDataSize(d);
nt.StartTimer()

// Send data to current application entity
pdu.Send(d,ae);

// Stop the timer, evaluate observed network bandwidth, and store it into ae
nt.EstimateCurrentBandwidth(ae);

```

**FIGURE 4**

```

//      It is assumed that ApplicationEntityList ael and DICOMObject d variables
//      have been already initialized based on the present network
//      configuration and the data to be sent.

// Compress DICOM object to accommodate current network bandwidth
int dsize = d.GetSize();
int ntime = ae.GetDownloadTime(); // maximum allowable download time
int ctime = dsize / ae.GetCurrentBandwidth(); //time we'll spend with current bandwidth
if(ctime > ntime) // low bandwidth, we need compression
{
    if(ae.CanAcceptLossy())      // we can use lossy compression
    {
        // Compress with maximum ratio allowed
        double ratio = min(ctime/ntime, ae.GetMaximumCompressionRatio());
        d.Compress(false,ratio);
    }
    else if(ae.CanAcceptCompressed()) // only lossless, try our best
    {
        d.Compress(true, /*ignored*/1.0);
    }
    else // no JPEG compression allowed, do nothing
    {
    }
}
else // network is fast enough, do not have to compress
{}

// Create PDU connection
PDU pdu;

// Send compressed data to current application entity
pdu.Send(d,ae);

```

**FIGURE 5**

```

class AccurateTimer
{
private :
    int Initialized;
    __int64 Frequency;
    __int64 BeginTime;
public :
    AccurateTimer()    // constructor
    {
        // get the frequency of the counter
        Initialized = QueryPerformanceFrequency( (LARGE_INTEGER
*)&Frequency );
    }

    void StartTimer()  // start timing
    {
        if( ! Initialized ) return 0; // error - couldn't get frequency
        // get the starting counter value
        QueryPerformanceCounter( (LARGE_INTEGER *)&BeginTime );
        return
    }

    double EndTimer()  // stop timing and get elapsed time in seconds
    {
        if( ! Initialized ) return 0.0; // error - couldn't get frequency
        // get the ending counter value
        __int64 endtime;
        QueryPerformanceCounter( (LARGE_INTEGER *)&endtime );
        // determine the elapsed counts
        __int64 elapsed = endtime - BeginTime;
        // convert counts to time in seconds and return it
        return (double)elapsed / (double)Frequency;
    }
}

```

**FIGURE 6**

```

bool SoundRecorder::Record(int sf, bool stereo, int max_time=300, int
max_size=1000000, int format=WAV)
{
    DWORD dwReturn;

    // Open a waveform-audio device with a new file for recording.
    if(!OpenMCI()) return;

    MCI_RECORD_PARMS mciRecordParms;
    // Set recording parameters here as fields in mciRecordParms
    mciRecordParms.dwFrom = 0;
    mciRecordParms.dwTo = max_time;

    ....
    mciRecordParms.dwCallback = (DWORD)(this->GetSafeHwnd());

    // Record
    if (dwReturn = mciSendCommand(m_Device.wDeviceID, MCI_RECORD,
MCI_FROM | MCI_NOTIFY, (DWORD)(LPVOID) &mciRecordParms))
    {
        AfxMessageBox(ErrorMCI(dwReturn, "Recording error: ");
        mciSendCommand(m_Device.wDeviceID, MCI_CLOSE, 0, NULL);
        return false;
    }
    return true;
}

```

**FIGURE 7**

```

void SoundRecorder::Insert(DICOMObject& dob)
{
    BYTE* sound;
    if(GetFormat() != MP3) // we have to encode the sound
    {
        SoundEncoder se;
        if(se.Encode(GetSound(), GetFormat(), sound)
        {
            return false; // we failed to encode
        }
    }
    // Store sound bytes into odb object
    ....
    return true;
}

```

**FIGURE 8**

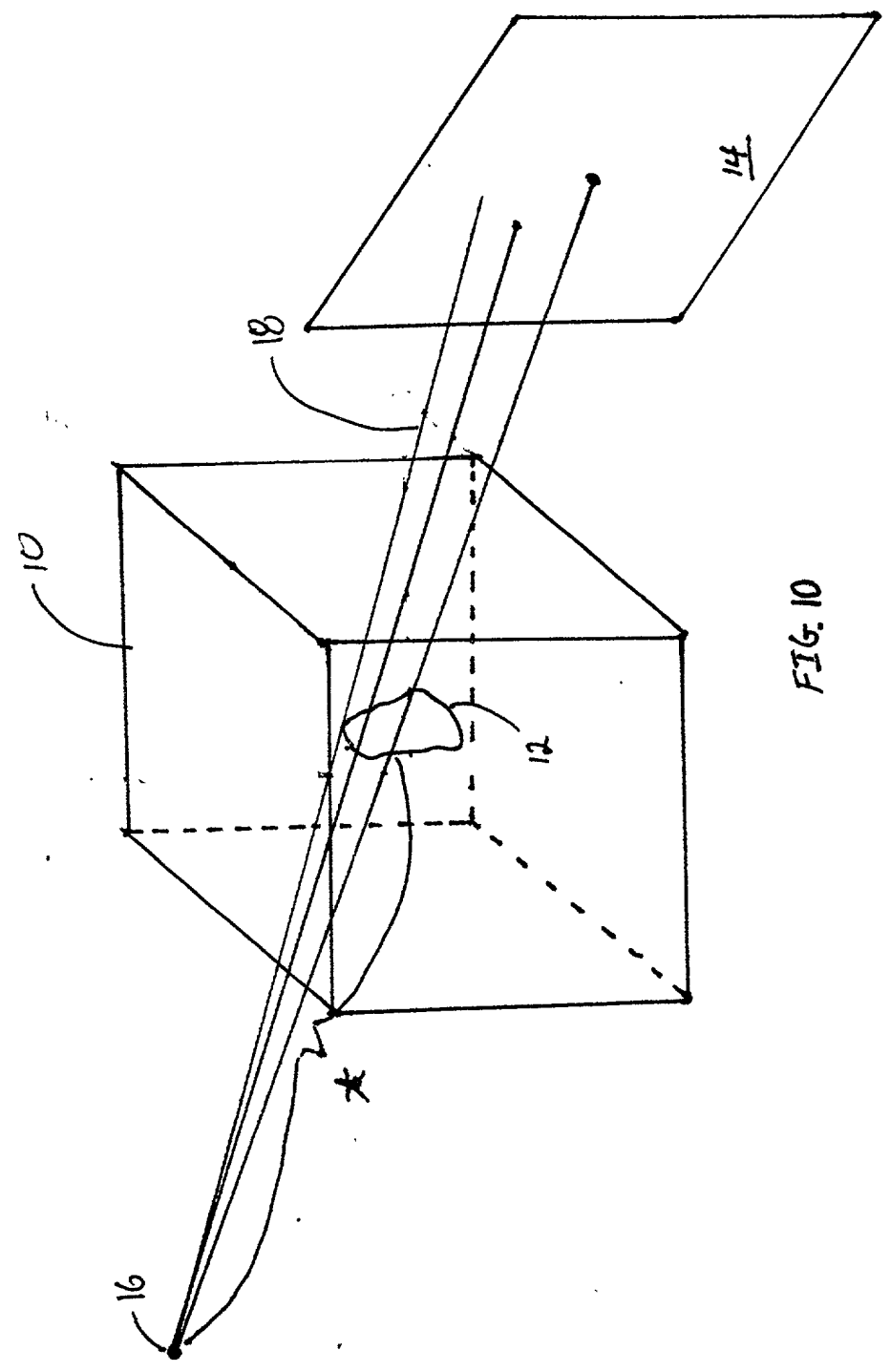
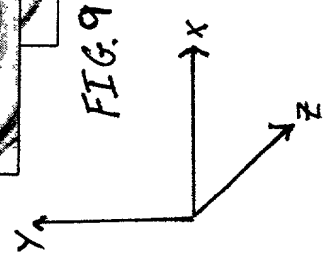
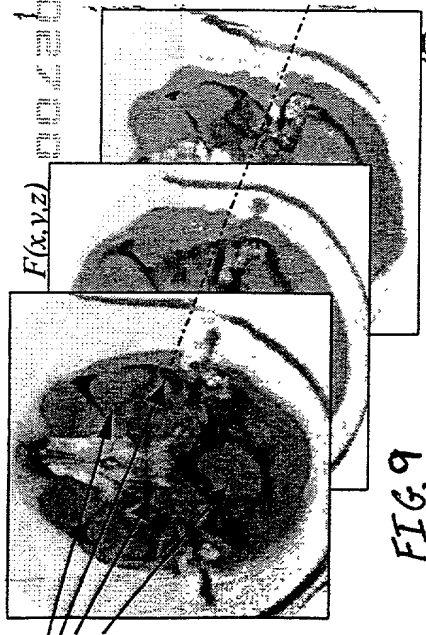


FIG. 10

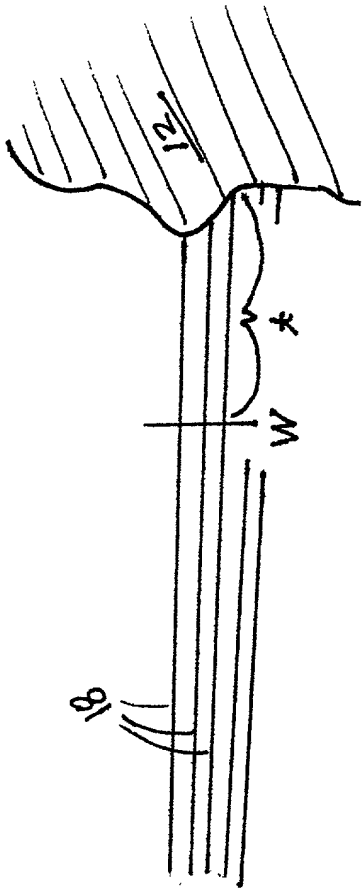


FIG. 11

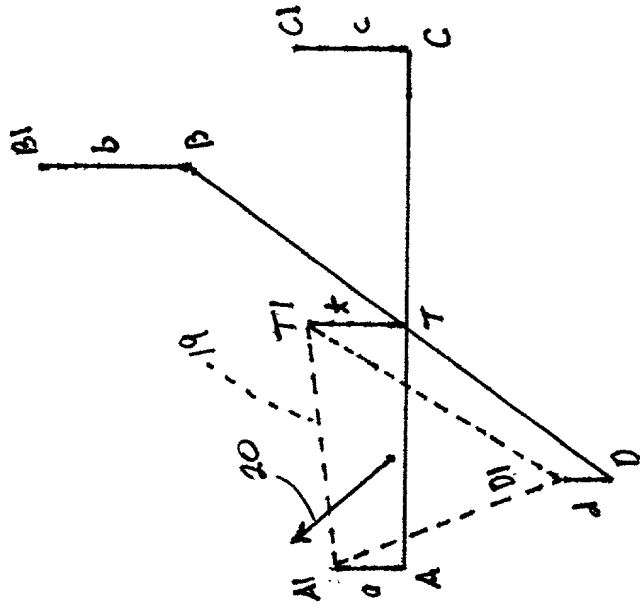


FIG. 12

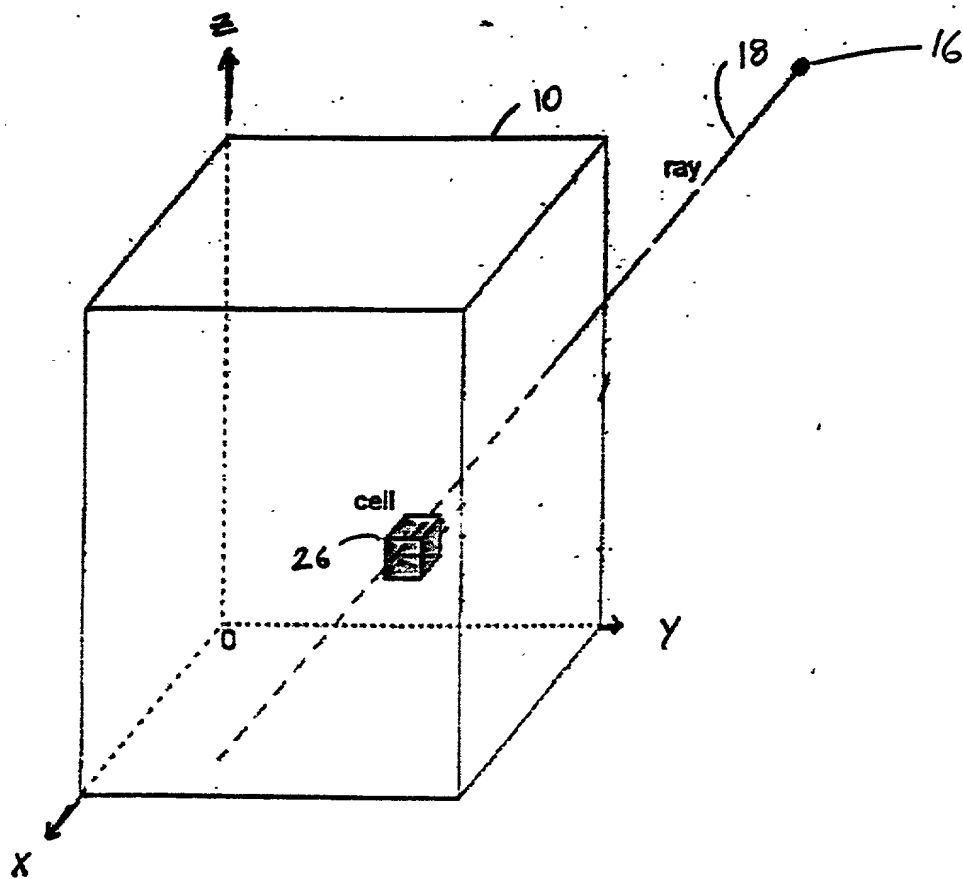


FIG. 13 (a)

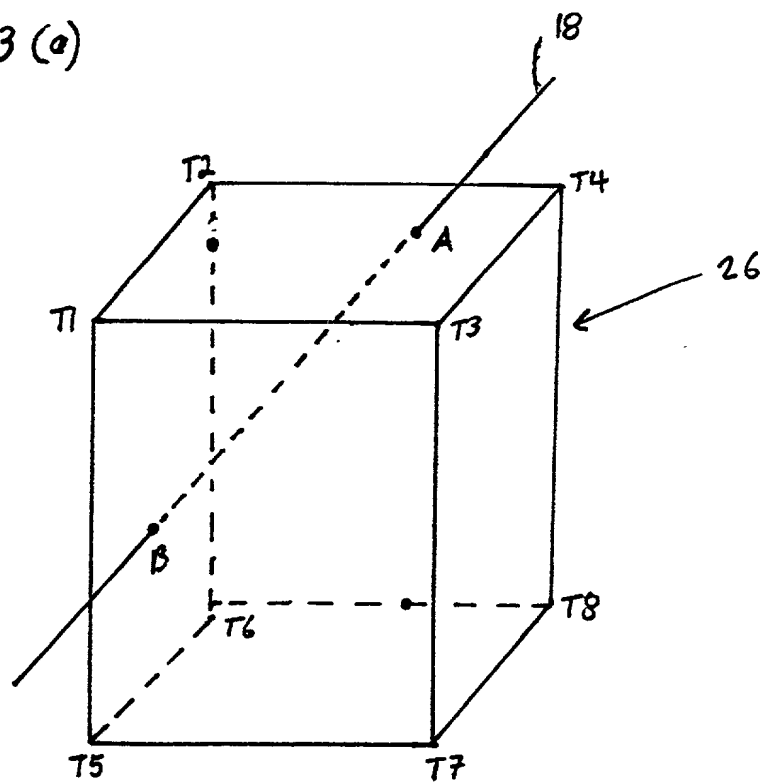


FIG. 13 (b)



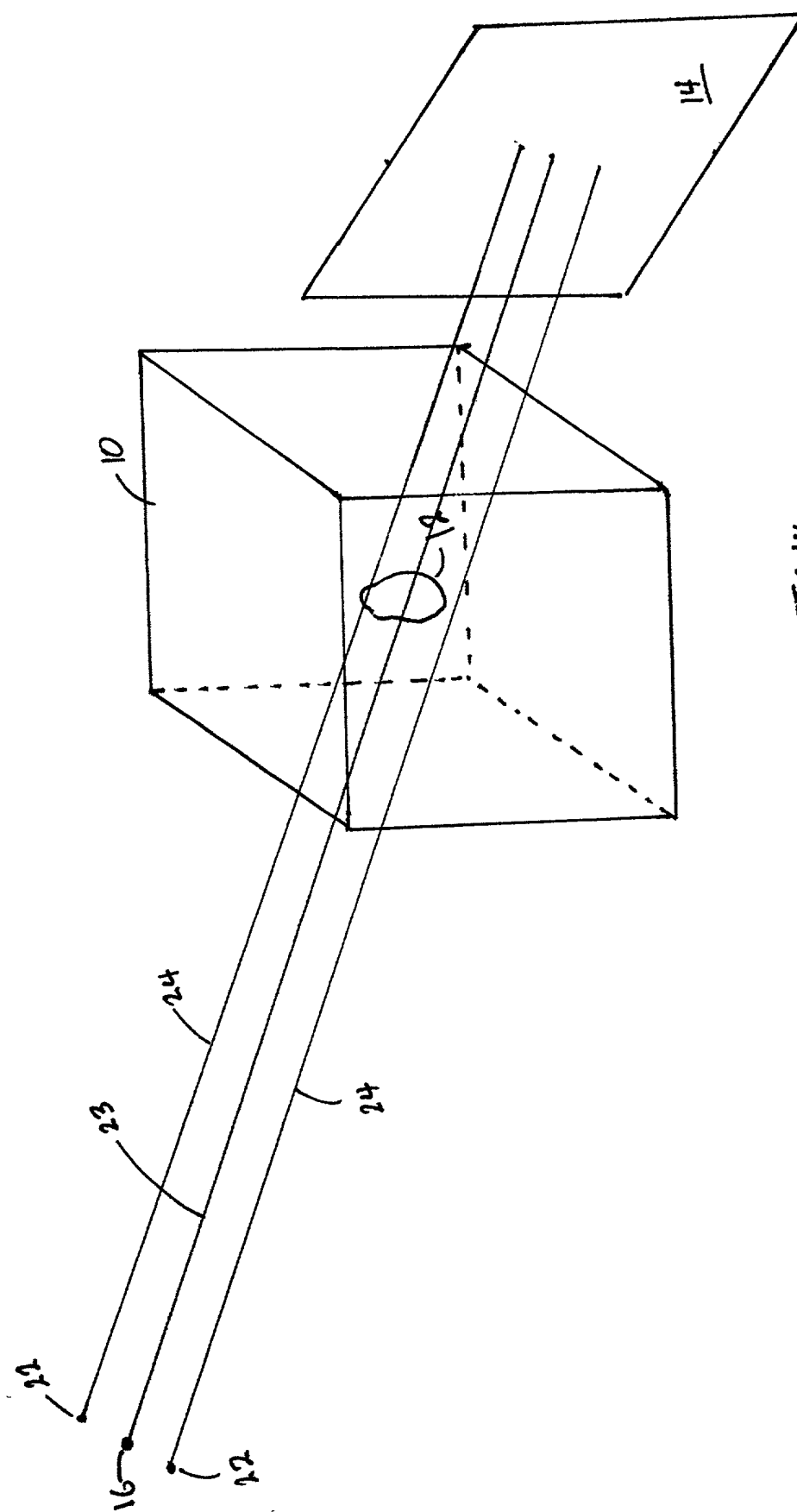
[illegible]

FIG. 14

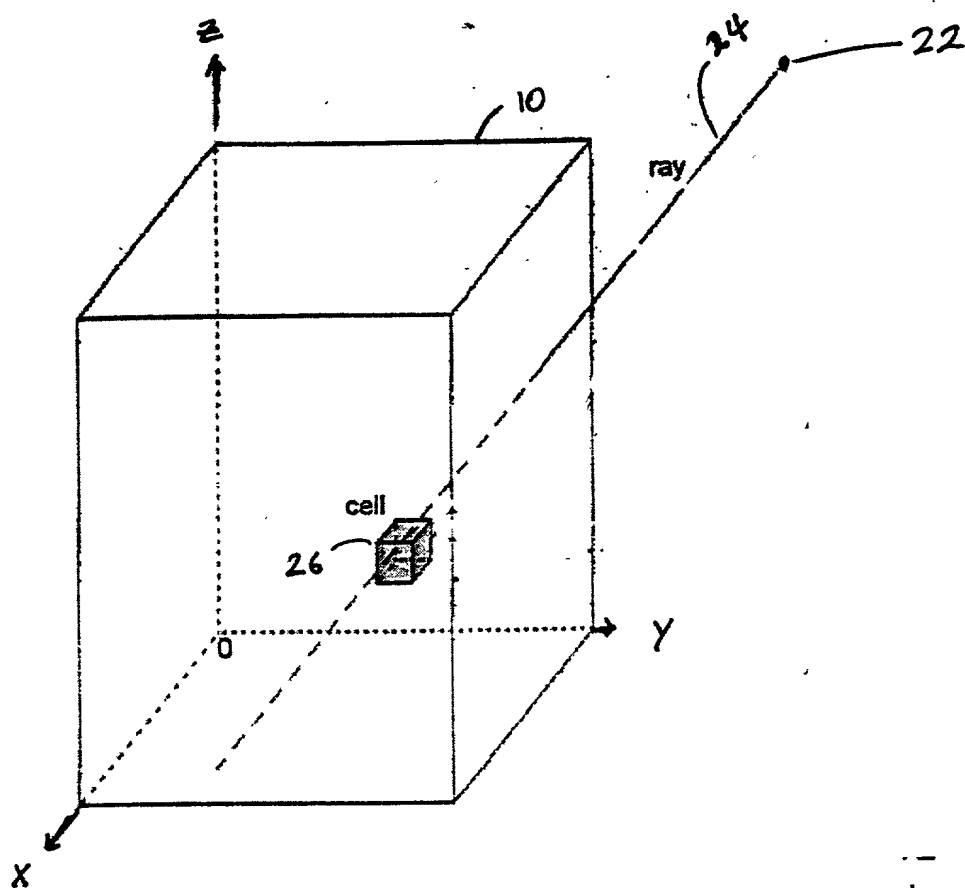


FIG. 15(a)

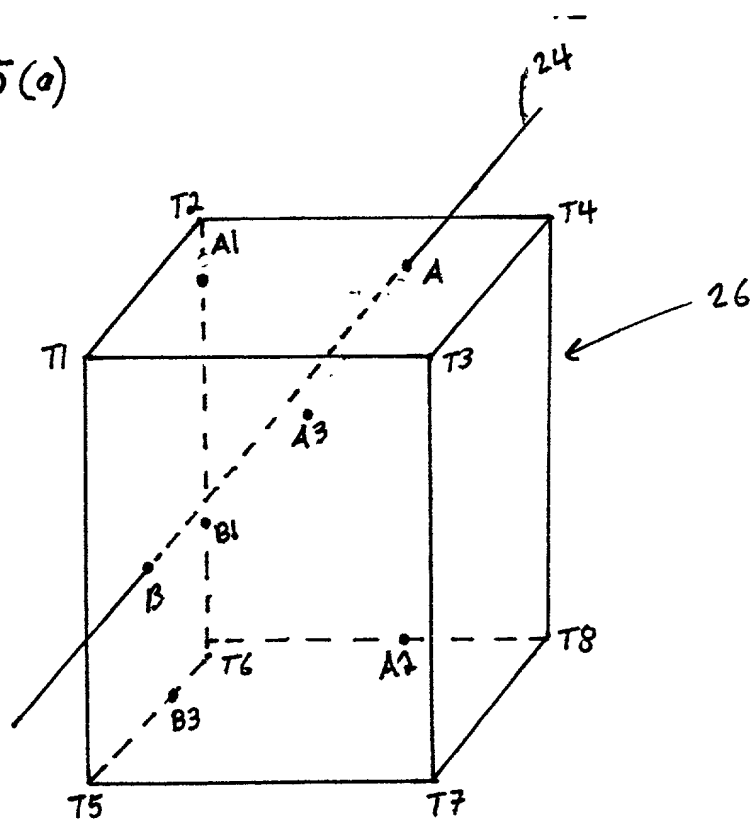


FIG. 15(b)

1 - linear root interpolation  
2 - parabolic root interpolation

FIG. 16

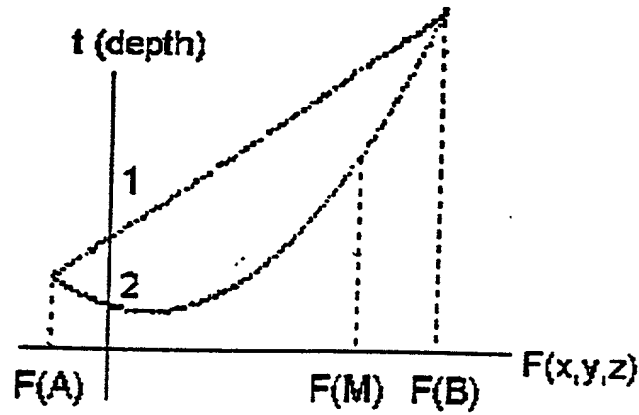


FIG. 17

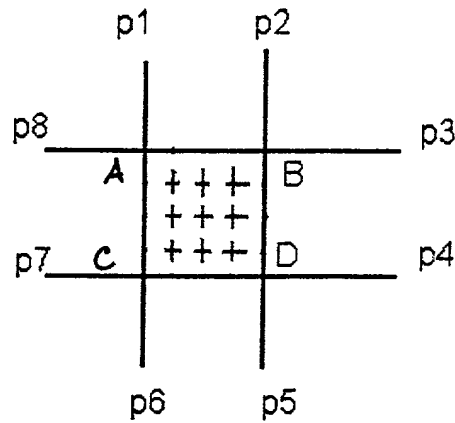
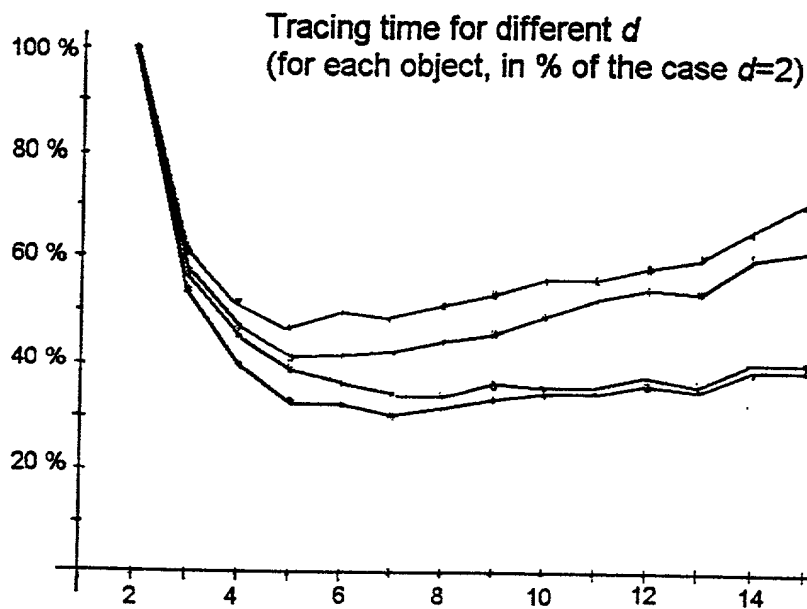


FIG. 18



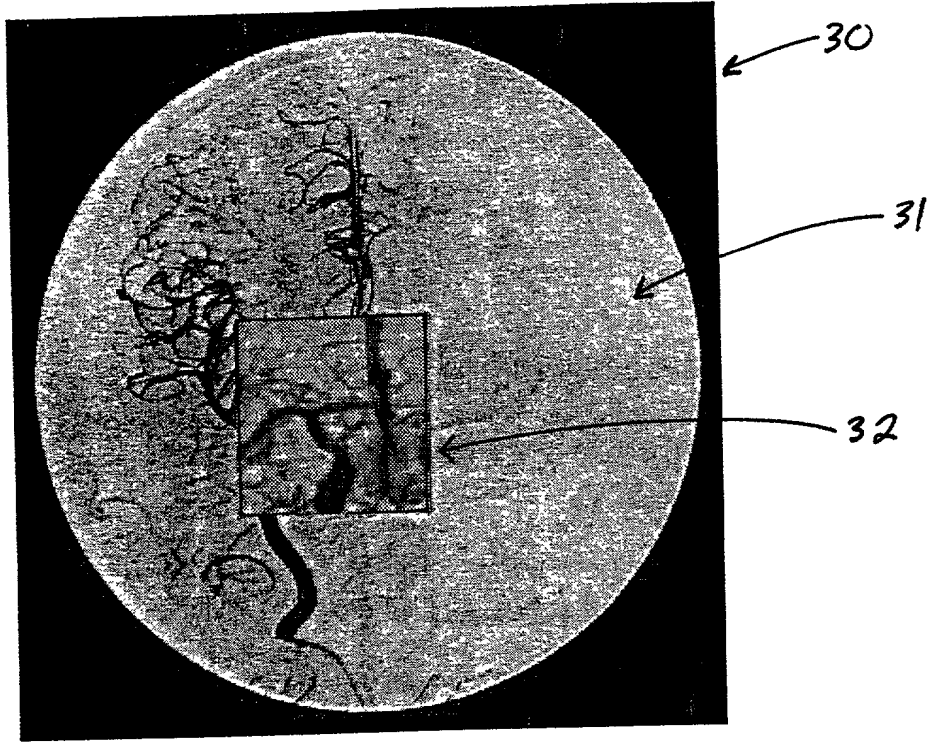


FIG. 19

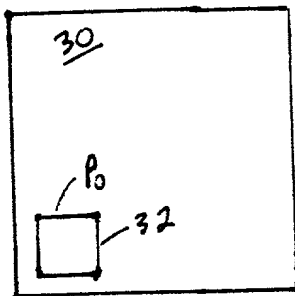


FIG. 20(a)

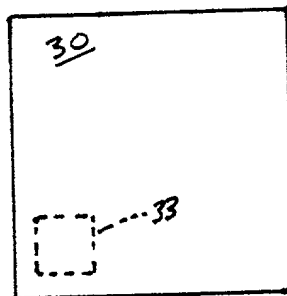


FIG. 20(b)

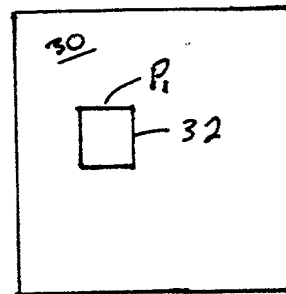


FIG. 20(c)

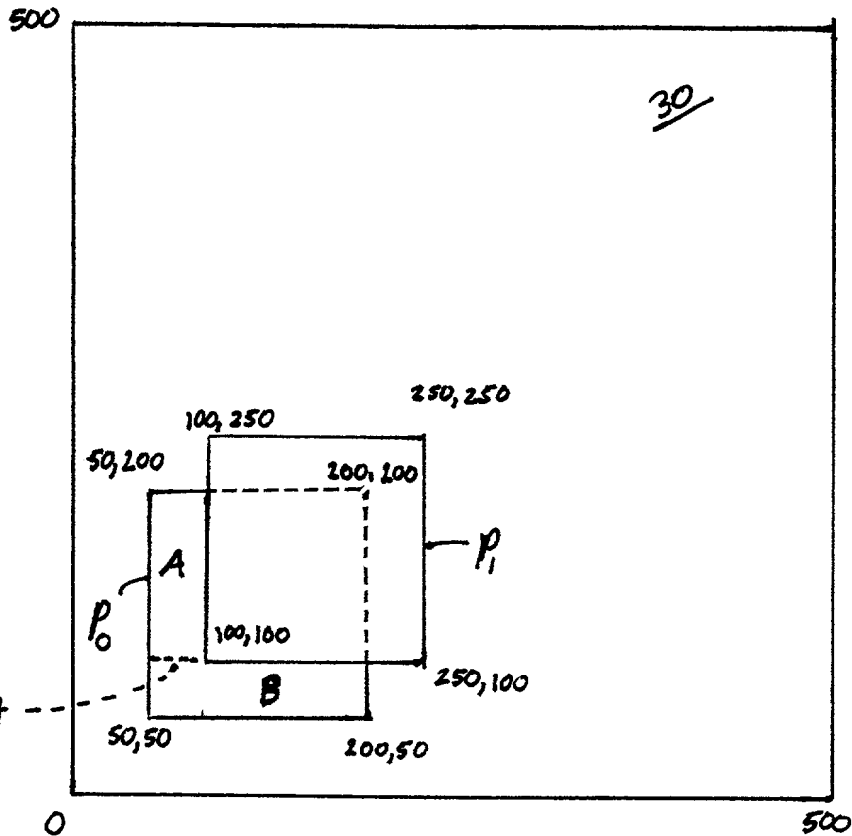


FIG. 21

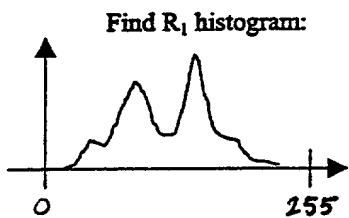


FIG. 23(a)

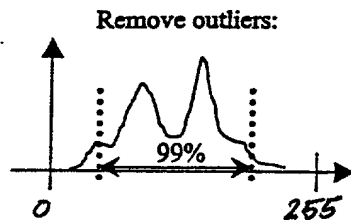


FIG. 23(b)

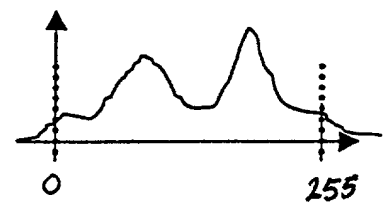


FIG. 23(c)

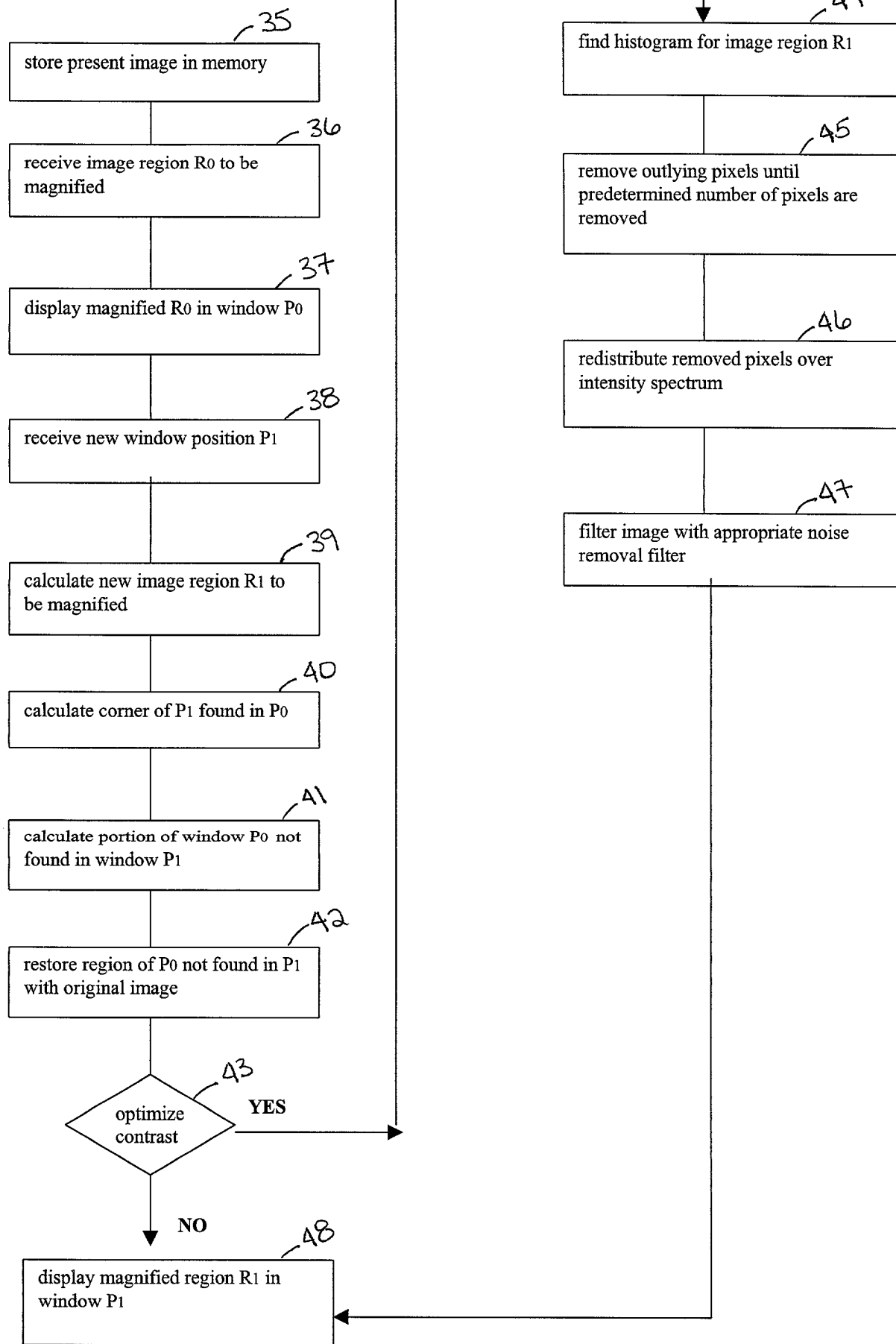


FIG. 22

**Inventor: Pianykh, et al.**

**Title: Radiologist Workstation**

**Atty. Doc. # 6451.064**

**Source Code Listing (Volume 1)**

**Inventor: Pianykh, et al.**

## Title: Radiologist Workstation

**Atty. Doc. # 6451.064**

## Source Code Listing (Volume 1)

```

/*
 * jcapimin.c
 *
 * Copyright (C) 1994-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains application interface code for the compression half
 * of the JPEG library. These are the "minimum" API routines that may be
 * needed in either the normal full-compression case or the transcoding-only
 * case.
 *
 * Most of the routines intended to be called directly by an application
 * are in this file or in jcapistd.c. But also see jcparam.c for
 * parameter-setup helper routines, jcomapi.c for routines shared by
 * compression and decompression, and jctrans.c for the transcoding case.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/*
 * Initialization of a JPEG compression object.
 * The error manager must already be set up (in case memory manager fails).
 */

GLOBAL(void)
jpeg_CreateCompress (j_compress_ptr cinfo, int version, size_t structsize)
{
  int i;

  /* Guard against version mismatches between library and caller. */
  cinfo->mem = NULL; /* so jpeg_destroy knows mem mgr not called */
  if (version != JPEG_LIB_VERSION)
    ERREXIT2(cinfo, JERR_BAD_LIB_VERSION, JPEG_LIB_VERSION, version);
  if (structsize != SIZEOF(struct jpeg_compress_struct))
    ERREXIT2(cinfo, JERR_BAD_STRUCT_SIZE,
      (int) SIZEOF(struct jpeg_compress_struct), (int) structsize);

  /* For debugging purposes, we zero the whole master structure.
   * But the application has already set the err pointer, and may have set
   * client_data, so we have to save and restore those fields.
   * Note: if application hasn't set client_data, tools like Purify may
   * complain here.
   */
  struct jpeg_error_mgr * err = cinfo->err;
  void * client_data = cinfo->client_data; /* ignore Purify complaint here */
  MEMZERO(cinfo, SIZEOF(struct jpeg_compress_struct));
  cinfo->err = err;
  cinfo->client_data = client_data;

  cinfo->is_decompressor = FALSE;

  /* Initialize a memory manager instance for this object */
  jinit_memory_mgr((j_common_ptr) cinfo);

  /* Zero out pointers to permanent structures. */
  cinfo->progress = NULL;
  cinfo->dest = NULL;

  cinfo->comp_info = NULL;

  for (i = 0; i < NUM_QUANT_TBLS; i++)
    cinfo->quant_tbl_ptrs[i] = NULL;

  for (i = 0; i < NUM_HUFF_TBLS; i++) {
    cinfo->dc_huff_tbl_ptrs[i] = NULL;
    cinfo->ac_huff_tbl_ptrs[i] = NULL;
  }

  cinfo->script_space = NULL;

  cinfo->input_gamma = 1.0; /* in case application forgets */

  /* OK, I'm ready */
  cinfo->global_state = CSTATE_START;
}

```



```

/*
 * Destruction of a JPEG compression object
 */

GLOBAL(void)
jpeg_destroy_compress (j_compress_ptr cinfo)
{
    jpeg_destroy((j_common_ptr) cinfo); /* use common routine */
}

/*
 * Abort processing of a JPEG compression operation,
 * but don't destroy the object itself.
 */

GLOBAL(void)
jpeg_abort_compress (j_compress_ptr cinfo)
{
    jpeg_abort((j_common_ptr) cinfo); /* use common routine */
}

/*
 * Forcibly suppress or un-suppress all quantization and Huffman tables.
 * Marks all currently defined tables as already written (if suppress)
 * or not written (if !suppress). This will control whether they get emitted
 * by a subsequent jpeg_start_compress call.
 *
 * This routine is exported for use by applications that want to produce
 * abbreviated JPEG datastreams. It logically belongs in jcparm.c, but
 * since it is called by jpeg_start_compress, we put it here --- otherwise
 * jcparm.o would be linked whether the application used it or not.
 */

GLOBAL(void)
jpeg_suppress_tables (j_compress_ptr cinfo, boolean suppress)
{
    int i;
    JQUANT_TBL * qtbl;
    JHUFF_TBL * htbl;

    for (i = 0; i < NUM_QUANT_TBLS; i++) {
        if ((qtbl = cinfo->quant_tbl_ptrs[i]) != NULL)
            qtbl->sent_table = suppress;
    }

    for (i = 0; i < NUM_HUFF_TBLS; i++) {
        if ((htbl = cinfo->dc_huff_tbl_ptrs[i]) != NULL)
            htbl->sent_table = suppress;
        if ((htbl = cinfo->ac_huff_tbl_ptrs[i]) != NULL)
            htbl->sent_table = suppress;
    }
}

/*
 * Finish JPEG compression.
 *
 * If a multipass operating mode was selected, this may do a great deal of
 * work including most of the actual output.
 */

GLOBAL(void)
jpeg_finish_compress (j_compress_ptr cinfo)
{
    JDIMENSION iMCU_row;

    if (cinfo->global_state == CSTATE_SCANNING ||
        cinfo->global_state == CSTATE_RAW_OK) {
        /* Terminate first pass */
        if (cinfo->next_scanline < cinfo->image_height)
            ERREXIT(cinfo, JERR_TOO_LITTLE_DATA);
        (*cinfo->master->finish_pass) (cinfo);
    } else if (cinfo->global_state != CSTATE_WRCOEFS)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
    /* Perform any remaining passes */
    while (! cinfo->master->is_last_pass) {

```

```

(*cinfo->master->prepare_for_pass) (cinfo);
for (iMCU_row = 0; iMCU_row < cinfo->total_iMCU_rows; iMCU_row++) {
    if (cinfo->progress != NULL) {
        cinfo->progress->pass_counter = (long) iMCU_row;
        cinfo->progress->pass_limit = (long) cinfo->total_iMCU_rows;
        (*cinfo->progress->progress_monitor) ((j_common_ptr) cinfo);
    }
    /* We bypass the main controller and invoke coef controller directly;
     * all work is being done from the coefficient buffer.
     */
    if (! (*cinfo->coef->compress_data) (cinfo, (JSAMPIMAGE) NULL))
        ERREXIT(cinfo, JERR_CANT_SUSPEND);
}
(*cinfo->master->finish_pass) (cinfo);
}
/* Write EOI, do final cleanup */
(*cinfo->marker->write_file_trailer) (cinfo);
(*cinfo->dest->term_destination) (cinfo);
/* We can use jpeg_abort to release memory and reset global_state */
jpeg_abort((j_common_ptr) cinfo);
}

/*
 * Write a special marker.
 * This is only recommended for writing COM or APPn markers.
 * Must be called after jpeg_start_compress() and before
 * first call to jpeg_write_scanlines() or jpeg_write_raw_data().
 */

GLOBAL(void)
jpeg_write_marker (j_compress_ptr cinfo, int marker,
                  const JOCTET *dataptr, unsigned int datalen)
{
    JMETHOD(void, write_marker_byte, (j_compress_ptr info, int val));

    if (cinfo->next_scanline != 0 ||
        (cinfo->global_state != CSTATE_SCANNING &&
         cinfo->global_state != CSTATE_RAW_OK &&
         cinfo->global_state != CSTATE_WRCOEFS))
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);

    (*cinfo->marker->write_marker_header) (cinfo, marker, datalen);
    write_marker_byte = cinfo->marker->write_marker_byte; /* copy for speed */
    while (datalen-- > 0) {
        (*write_marker_byte) (cinfo, *dataptr);
        dataptr++;
    }

    /* Same, but piecemeal. */

GLOBAL(void)
jpeg_write_m_header (j_compress_ptr cinfo, int marker, unsigned int datalen)
{
    if (cinfo->next_scanline != 0 ||
        (cinfo->global_state != CSTATE_SCANNING &&
         cinfo->global_state != CSTATE_RAW_OK &&
         cinfo->global_state != CSTATE_WRCOEFS))
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);

    (*cinfo->marker->write_marker_header) (cinfo, marker, datalen);
}

GLOBAL(void)
jpeg_write_m_byte (j_compress_ptr cinfo, int val)
{
    (*cinfo->marker->write_marker_byte) (cinfo, val);
}

/*
 * Alternate compression function: just write an abbreviated table file.
 * Before calling this, all parameters and a data destination must be set up.
 *
 * To produce a pair of files containing abbreviated tables and abbreviated
 * image data, one would proceed as follows:
 *
 *     initialize JPEG object
 *     set JPEG parameters

```

```

*      set destination to table file
*      jpeg_write_tables(cinfo);
*      set destination to image file
*      jpeg_start_compress(cinfo, FALSE);
*      write data...
*      jpeg_finish_compress(cinfo);
*
* jpeg_write_tables has the side effect of marking all tables written
* (same as jpeg_suppress_tables(..., TRUE)). Thus a subsequent start_compress
* will not re-emit the tables unless it is passed write_all_tables=TRUE.
*/

```

```

GLOBAL(void)
jpeg_write_tables (j_compress_ptr cinfo)
{
    if (cinfo->global_state != CSTATE_START)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);

    /* (Re)initialize error mgr and destination modules */
    (*cinfo->err->reset_error_mgr) ((j_common_ptr) cinfo);
    (*cinfo->dest->init_destination) (cinfo);
    /* Initialize the marker writer ... bit of a crock to do it here. */
    jinit_marker_writer(cinfo);
    /* Write them tables! */
    (*cinfo->marker->write_tables_only) (cinfo);
    /* And clean up. */
    (*cinfo->dest->term_destination) (cinfo);
    /*
     * In library releases up through v6a, we called jpeg_abort() here to free
     * any working memory allocated by the destination manager and marker
     * writer. Some applications had a problem with that: they allocated space
     * of their own from the library memory manager, and didn't want it to go
     * away during write_tables. So now we do nothing. This will cause a
     * memory leak if an app calls write_tables repeatedly without doing a full
     * compression cycle or otherwise resetting the JPEG object. However, that
     * seems less bad than unexpectedly freeing memory in the normal case.
     * An app that prefers the old behavior can call jpeg_abort for itself after
     * each call to jpeg_write_tables().
     */
}

```

```

/*
 * jcapistd.c
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains application interface code for the compression half
 * of the JPEG library. These are the "standard" API routines that are
 * used in the normal full-compression case. They are not used by a
 * transcoding-only application. Note that if an application links in
 * jpeg_start_compress, it will end up linking in the entire compressor.
 * We thus must separate this file from jcapimin.c to avoid linking the
 * whole compression library into a transcoder.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/*
 * Compression initialization.
 * Before calling this, all parameters and a data destination must be set up.
 *
 * We require a write_all_tables parameter as a failsafe check when writing
 * multiple datastreams from the same compression object. Since prior runs
 * will have left all the tables marked sent_table=TRUE, a subsequent run
 * would emit an abbreviated stream (no tables) by default. This may be what
 * is wanted, but for safety's sake it should not be the default behavior:
 * programmers should have to make a deliberate choice to emit abbreviated
 * images. Therefore the documentation and examples should encourage people
 * to pass write_all_tables=TRUE; then it will take active thought to do the
 * wrong thing.
 */

GLOBAL(void)
jpeg_start_compress (j_compress_ptr cinfo, boolean write_all_tables)
{
    if (cinfo->global_state != CSTATE_START)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);

    if (write_all_tables)
        jpeg_suppress_tables(cinfo, FALSE); /* mark all tables to be written */

    /* (Re)initialize error mgr and destination modules */
    (*cinfo->err->reset_error_mgr) ((j_common_ptr) cinfo);
    (*cinfo->dest->init_destination) (cinfo);
    /* Perform master selection of active modules */
    jinit_compress_master(cinfo);
    /* Set up for the first pass */
    (*cinfo->master->prepare_for_pass) (cinfo);
    /* Ready for application to drive first pass through jpeg_write_scanlines
     * or jpeg_write_raw_data.
     */
    cinfo->next_scanline = 0;
    cinfo->global_state = (cinfo->raw_data_in ? CSTATE_RAW_OK : CSTATE_SCANNING);
}

/*
 * Write some scanlines of data to the JPEG compressor.
 *
 * The return value will be the number of lines actually written.
 * This should be less than the supplied num_lines only in case that
 * the data destination module has requested suspension of the compressor,
 * or if more than image_height scanlines are passed in.
 *
 * Note: we warn about excess calls to jpeg_write_scanlines() since
 * this likely signals an application programmer error. However,
 * excess scanlines passed in the last valid call are *silently* ignored,
 * so that the application need not adjust num_lines for end-of-image
 * when using a multiple-scanline buffer.
 */

GLOBAL(JDIMENSION)
jpeg_write_scanlines (j_compress_ptr cinfo, JSAMPARRAY scanlines,
                     JDIMENSION num_lines)
{
    JDIMENSION row_ctr, rows_left;

```

```

if (cinfo->global_state != CSTATE_SCANNING)
    ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
if (cinfo->next_scanline >= cinfo->image_height)
    WARNMS(cinfo, JWRN_TOO_MUCH_DATA);

/* Call progress monitor hook if present */
if (cinfo->progress != NULL) {
    cinfo->progress->pass_counter = (long) cinfo->next_scanline;
    cinfo->progress->pass_limit = (long) cinfo->image_height;
    (*cinfo->progress->progress_monitor) ((j_common_ptr) cinfo);
}

/* Give master control module another chance if this is first call to
 * jpeg_write_scanlines. This lets output of the frame/scan headers be
 * delayed so that application can write COM, etc, markers between
 * jpeg_start_compress and jpeg_write_scanlines.
 */
if (cinfo->master->call_pass_startup)
    (*cinfo->master->pass_startup) (cinfo);

/* Ignore any extra scanlines at bottom of image. */
rows_left = cinfo->image_height - cinfo->next_scanline;
if (num_lines > rows_left)
    num_lines = rows_left;

row_ctr = 0;
(*cinfo->main->process_data) (cinfo, scanlines, &row_ctr, num_lines);
cinfo->next_scanline += row_ctr;
return row_ctr;
}

/*
 * Alternate entry point to write raw data.
 * Processes exactly one iMCU row per call, unless suspended.
 */
GLOBAL(JDIMENSION)
jpeg_write_raw_data (j_compress_ptr cinfo, JSAMPIMAGE data,
                    JDIMENSION num_lines)
{
    JDIMENSION lines_per_iMCU_row;

    if (cinfo->global_state != CSTATE_RAW_OK)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
    if (cinfo->next_scanline >= cinfo->image_height) {
        WARNMS(cinfo, JWRN_TOO_MUCH_DATA);
        return 0;
    }

    /* Call progress monitor hook if present */
    if (cinfo->progress != NULL) {
        cinfo->progress->pass_counter = (long) cinfo->next_scanline;
        cinfo->progress->pass_limit = (long) cinfo->image_height;
        (*cinfo->progress->progress_monitor) ((j_common_ptr) cinfo);
    }

    /* Give master control module another chance if this is first call to
     * jpeg_write_raw_data. This lets output of the frame/scan headers be
     * delayed so that application can write COM, etc, markers between
     * jpeg_start_compress and jpeg_write_raw_data.
     */
    if (cinfo->master->call_pass_startup)
        (*cinfo->master->pass_startup) (cinfo);

    /* Verify that at least one iMCU row has been passed. */
    lines_per_iMCU_row = cinfo->max_v_samp_factor * DCTSIZE;
    if (num_lines < lines_per_iMCU_row)
        ERREXIT(cinfo, JERR_BUFFER_SIZE);

    /* Directly compress the row. */
    if (! (*cinfo->coef->compress_data) (cinfo, data)) {
        /* If compressor did not consume the whole row, suspend processing. */
        return 0;
    }

    /* OK, we processed one iMCU row. */
    cinfo->next_scanline += lines_per_iMCU_row;
    return lines_per_iMCU_row;
}

```

3

```

/*
 * jccoefct.c
 *
 * Copyright (C) 1994-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains the coefficient buffer controller for compression.
 * This controller is the top level of the JPEG compressor proper.
 * The coefficient buffer lies between forward-DCT and entropy encoding steps.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/* We use a full-image coefficient buffer when doing Huffman optimization,
 * and also for writing multiple-scan JPEG files. In all cases, the DCT
 * step is run during the first pass, and subsequent passes need only read
 * the buffered coefficients.
 */

#ifdef ENTROPY_OPT_SUPPORTED
#define FULL_COEF_BUFFER_SUPPORTED
#else
#ifdef C_MULTISCAN_FILES_SUPPORTED
#define FULL_COEF_BUFFER_SUPPORTED
#endif
#endif

/* Private buffer controller object */
typedef struct {
  struct jpeg_c_coef_controller pub; /* public fields */

  JDIMENSION iMCU_row_num; /* iMCU row # within image */
  JDIMENSION mcu_ctr; /* counts MCUs processed in current row */
  int MCU_vert_offset; /* counts MCU rows within iMCU row */
  int MCU_rows_per_iMCU_row; /* number of such rows needed */

  /* For single-pass compression, it's sufficient to buffer just one MCU
   * (although this may prove a bit slow in practice). We allocate a
   * workspace of C_MAX_BLOCKS_IN_MCU coefficient blocks, and reuse it for each
   * MCU constructed and sent. (On 80x86, the workspace is FAR even though
   * it's not really very big; this is to keep the module interfaces unchanged
   * when a large coefficient buffer is necessary.)
   * In multi-pass modes, this array points to the current MCU's blocks
   * within the virtual arrays.
   */
  JBLOCKROW MCU_buffer[C_MAX_BLOCKS_IN_MCU];

  /* In multi-pass modes, we need a virtual block array for each component. */
  jvirt_barray_ptr whole_image[MAX_COMPONENTS];
} my_coef_controller;

typedef my_coef_controller * my_coef_ptr;

/* Forward declarations */
METHODDEF(boolean) compress_data
  JPP((j_compress_ptr cinfo, JSAMPIMAGE input_buf));
#ifdef FULL_COEF_BUFFER_SUPPORTED
METHODDEF(boolean) compress_first_pass
  JPP((j_compress_ptr cinfo, JSAMPIMAGE input_buf));
METHODDEF(boolean) compress_output
  JPP((j_compress_ptr cinfo, JSAMPIMAGE input_buf));
#endif

LOCAL(void)
start_iMCU_row (j_compress_ptr cinfo)
/* Reset within-iMCU-row counters for a new row */
{
  my_coef_ptr coef = (my_coef_ptr) cinfo->coef;

  /* In an interleaved scan, an MCU row is the same as an iMCU row.
   * In a noninterleaved scan, an iMCU row has v_samp_factor MCU rows.
   * But at the bottom of the image, process only what's left.
   */

```

```

if (cinfo->comps_in_scan > 1) {
    coef->MCU_rows_per_iMCU_row = 1;
} else {
    if (coef->iMCU_row_num < (cinfo->total_iMCU_rows-1))
        coef->MCU_rows_per_iMCU_row = cinfo->cur_comp_info[0]->v_samp_factor;
    else
        coef->MCU_rows_per_iMCU_row = cinfo->cur_comp_info[0]->last_row_height;
}

coef->mcu_ctr = 0;
coef->MCU_vert_offset = 0;
}

```

```

/*
 * Initialize for a processing pass.
 */

```

```

METHODDEF(void)
start_pass_coef (j_compress_ptr cinfo, J_BUF_MODE pass_mode)

```

```

{
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;

    coef->iMCU_row_num = 0;
    start_iMCU_row(cinfo);

    switch (pass_mode) {
    case JBUF_PASS_THRU:
        if (coef->whole_image[0] != NULL)
            ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
        coef->pub.compress_data = compress_data;
        break;
#ifdef FULL_COEF_BUFFER_SUPPORTED
    case JBUF_SAVE_AND_PASS:
        if (coef->whole_image[0] == NULL)
            ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
        coef->pub.compress_data = compress_first_pass;
        break;
    case JBUF_CRANK_DEST:
        if (coef->whole_image[0] == NULL)
            ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
        coef->pub.compress_data = compress_output;
        break;
    #endif
    default:
        ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
        break;
    }
}

```

```

/*
 * Process some data in the single-pass case.
 * We process the equivalent of one fully interleaved MCU row ("iMCU" row)
 * per call, ie, v_samp_factor block rows for each component in the image.
 * Returns TRUE if the iMCU row is completed, FALSE if suspended.
 *
 * NB: input_buf contains a plane for each component in image,
 * which we index according to the component's SOF position.
 */

```

```

METHODDEF(boolean)
compress_data (j_compress_ptr cinfo, JSAMPIMAGE input_buf)

```

```

{
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;
    JDIMENSION MCU_col_num; /* index of current MCU within row */
    JDIMENSION last_MCU_col = cinfo->MCUs_per_row - 1;
    JDIMENSION last_iMCU_row = cinfo->total_iMCU_rows - 1;
    int blkn, bi, ci, yindex, yoffset, blockcnt;
    JDIMENSION ypos, xpos;
    jpeg_component_info *comp_ptr;

    /* Loop to write as much as one whole iMCU row */
    for (yoffset = coef->MCU_vert_offset; yoffset < coef->MCU_rows_per_iMCU_row;
         yoffset++) {
        for (MCU_col_num = coef->mcu_ctr; MCU_col_num <= last_MCU_col;
             MCU_col_num++) {
            /* Determine where data comes from in input_buf and do the DCT thing.
             * Each call on forward_DCT processes a horizontal row of DCT blocks
             * as wide as an MCU; we rely on having allocated the MCU_buffer[] blocks
            */

```



```

* sequentially. Dummy blocks at the right or bottom edge are filled in
* specially. The data in them does not matter for image reconstruction,
* so we fill them with values that will encode to the smallest amount of
* data, viz: all zeroes in the AC entries, DC entries equal to previous
* block's DC value. (Thanks to Thomas Kinsman for this idea.)
*/
blkcn = 0;
for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
    compptr = cinfo->cur_comp_info[ci];
    blockcnt = (MCU_col_num < last_MCU_col) ? compptr->MCU_width
        : compptr->last_col_width;
    xpos = MCU_col_num * compptr->MCU_sample_width;
    ypos = yoffset * DCTSIZE; /* ypos == (yoffset+yindex) * DCTSIZE */
    for (yindex = 0; yindex < compptr->MCU_height; yindex++) {
        if (coef->iMCU_row_num < last_iMCU_row ||
            yoffset+yindex < compptr->last_row_height) {
            (*cinfo->fdct->forward_DCT) (cinfo, compptr,
                input_buf[compptr->component_index],
                coef->MCU_buffer[blkcn],
                ypos, xpos, (JDIMENSION) blockcnt);
            if (blockcnt < compptr->MCU_width) {
                /* Create some dummy blocks at the right edge of the image. */
                jzero_far((void *) coef->MCU_buffer[blkcn + blockcnt],
                    (compptr->MCU_width - blockcnt) * SIZEOF(JBLOCK));
                for (bi = blockcnt; bi < compptr->MCU_width; bi++) {
                    coef->MCU_buffer[blkcn+bi][0][0] = coef->MCU_buffer[blkcn+bi-1][0][0];
                }
            }
        } else {
            /* Create a row of dummy blocks at the bottom of the image. */
            jzero_far((void *) coef->MCU_buffer[blkcn],
                compptr->MCU_width * SIZEOF(JBLOCK));
            for (bi = 0; bi < compptr->MCU_width; bi++) {
                coef->MCU_buffer[blkcn+bi][0][0] = coef->MCU_buffer[blkcn-1][0][0];
            }
        }
        blkcn += compptr->MCU_width;
        ypos += DCTSIZE;
    }
}
/* Try to write the MCU. In event of a suspension failure, we will
* re-DCT the MCU on restart (a bit inefficient, could be fixed...)
*/
if (! (*cinfo->entropy->encode_mcu) (cinfo, coef->MCU_buffer)) {
    /* Suspension forced; update state counters and exit */
    coef->MCU_vert_offset = yoffset;
    coef->mcu_ctr = MCU_col_num;
    return FALSE;
}
/* Completed an MCU row, but perhaps not an iMCU row */
coef->mcu_ctr = 0;
/* Completed the iMCU row, advance counters for next one */
coef->iMCU_row_num++;
start_iMCU_row(cinfo);
return TRUE;
}

#ifdef FULL_COEF_BUFFER_SUPPORTED
/*
* Process some data in the first pass of a multi-pass case.
* We process the equivalent of one fully interleaved MCU row ("iMCU" row)
* per call, ie, v_samp_factor block rows for each component in the image.
* This amount of data is read from the source buffer, DCT'd and quantized,
* and saved into the virtual arrays. We also generate suitable dummy blocks
* as needed at the right and lower edges. (The dummy blocks are constructed
* in the virtual arrays, which have been padded appropriately.) This makes
* it possible for subsequent passes not to worry about real vs. dummy blocks.
*
* We must also emit the data to the entropy encoder. This is conveniently
* done by calling compress_output() after we've loaded the current strip
* of the virtual arrays.
*
* NB: input_buf contains a plane for each component in image. All
* components are DCT'd and loaded into the virtual arrays in this pass.
* However, it may be that only a subset of the components are emitted to
* the entropy encoder during this first pass; be careful about looking

```

```

* at the scan-dependent variables (MCU dimensions, etc).
*/

```

```

METHODDEF(boolean)

```

```

compress_first_pass (j_compress_ptr cinfo, JSAMPIMAGE input_buf)

```

```

{
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;
    JDIMENSION last_iMCU_row = cinfo->total_iMCU_rows - 1;
    JDIMENSION blocks_across, MCUs_across, MCUindex;
    int bi, ci, h_samp_factor, block_row, block_rows, ndummy;
    JCOEF lastDC;
    jpeg_component_info *comp_ptr;
    JBLOCKARRAY buffer;
    JBLOCKROW thisblockrow, lastblockrow;

    for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
        ci++, comp_ptr++) {
        /* Align the virtual buffer for this component. */
        buffer = (*cinfo->mem->access_virt_barray)
            ((j_common_ptr) cinfo, coef->whole_image[ci],
             coef->iMCU_row_num * comp_ptr->v_samp_factor,
             (JDIMENSION) comp_ptr->v_samp_factor, TRUE);
        /* Count non-dummy DCT block rows in this iMCU row. */
        if (coef->iMCU_row_num < last_iMCU_row)
            block_rows = comp_ptr->v_samp_factor;
        else {
            /* NB: can't use last_row_height here, since may not be set! */
            block_rows = (int) (comp_ptr->height_in_blocks % comp_ptr->v_samp_factor);
            if (block_rows == 0) block_rows = comp_ptr->v_samp_factor;
        }
        blocks_across = comp_ptr->width_in_blocks;
        h_samp_factor = comp_ptr->h_samp_factor;
        /* Count number of dummy blocks to be added at the right margin. */
        ndummy = (int) (blocks_across % h_samp_factor);
        if (ndummy > 0)
            ndummy = h_samp_factor - ndummy;
        /* Perform DCT for all non-dummy blocks in this iMCU row. Each call
         * on forward_DCT processes a complete horizontal row of DCT blocks.
         */
        for (block_row = 0; block_row < block_rows; block_row++) {
            thisblockrow = buffer[block_row];
            (*cinfo->fdct->forward_DCT) (cinfo, comp_ptr,
                                         input_buf[ci], thisblockrow,
                                         (JDIMENSION) (block_row * DCTSIZE),
                                         (JDIMENSION) 0, blocks_across);
            if (ndummy > 0) {
                /* Create dummy blocks at the right edge of the image. */
                thisblockrow += blocks_across; /* => first dummy block */
                jzero_far((void *) thisblockrow, ndummy * SIZEOF(JBLOCK));
                lastDC = thisblockrow[-1][0];
                for (bi = 0; bi < ndummy; bi++) {
                    thisblockrow[bi][0] = lastDC;
                }
            }
        }
        /* If at end of image, create dummy block rows as needed.
         * The tricky part here is that within each MCU, we want the DC values
         * of the dummy blocks to match the last real block's DC value.
         * This squeezes a few more bytes out of the resulting file...
         */
        if (coef->iMCU_row_num == last_iMCU_row) {
            blocks_across += ndummy; /* include lower right corner */
            MCUs_across = blocks_across / h_samp_factor;
            for (block_row = block_rows; block_row < comp_ptr->v_samp_factor;
                block_row++) {
                thisblockrow = buffer[block_row];
                lastblockrow = buffer[block_row-1];
                jzero_far((void *) thisblockrow,
                         (size_t) (blocks_across * SIZEOF(JBLOCK)));
                for (MCUindex = 0; MCUindex < MCUs_across; MCUindex++) {
                    lastDC = lastblockrow[h_samp_factor-1][0];
                    for (bi = 0; bi < h_samp_factor; bi++) {
                        thisblockrow[bi][0] = lastDC;
                    }
                    thisblockrow += h_samp_factor; /* advance to next MCU in row */
                    lastblockrow += h_samp_factor;
                }
            }
        }
    }
}

```

```

/* NB: compress_output will increment iMCU_row_num if successful.
 * A suspension return will result in redoing all the work above next time.
 */

/* Emit data to the entropy encoder, sharing code with subsequent passes */
return compress_output(cinfo, input_buf);
}

/*
 * Process some data in subsequent passes of a multi-pass case.
 * We process the equivalent of one fully interleaved MCU row ("iMCU" row)
 * per call, ie, v_samp_factor block rows for each component in the scan.
 * The data is obtained from the virtual arrays and fed to the entropy coder.
 * Returns TRUE if the iMCU row is completed, FALSE if suspended.
 *
 * NB: input_buf is ignored; it is likely to be a NULL pointer.
 */

METHODDEF(boolean)
compress_output (j_compress_ptr cinfo, JSAMPIMAGE input_buf)
{
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;
    JDIMENSION MCU_col_num; /* index of current MCU within row */
    int blkcn, ci, xindex, yindex, yoffset;
    JDIMENSION start_col;
    JBLOCKARRAY buffer[MAX_COMPS_IN_SCAN];
    JBLOCKROW buffer_ptr;
    jpeg_component_info *comp_ptr;

    /* Align the virtual buffers for the components used in this scan.
     * NB: during first pass, this is safe only because the buffers will
     * already be aligned properly, so jmemmgr.c won't need to do any I/O.
     */
    for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
        comp_ptr = cinfo->cur_comp_info[ci];
        buffer[ci] = (*cinfo->mem->access_virt_barray)
            ((j_common_ptr) cinfo, coef->whole_image[comp_ptr->component_index],
             coef->iMCU_row_num * comp_ptr->v_samp_factor,
             (JDIMENSION) comp_ptr->v_samp_factor, FALSE);
    }

    /* Loop to process one whole iMCU row */
    for (yoffset = coef->MCU_vert_offset; yoffset < coef->MCU_rows_per_iMCU_row;
         yoffset++) {
        for (MCU_col_num = coef->mcu_ctr; MCU_col_num < cinfo->MCUs_per_row;
             MCU_col_num++) {
            /* Construct list of pointers to DCT blocks belonging to this MCU */
            blkcn = 0; /* index of current DCT block within MCU */
            for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
                comp_ptr = cinfo->cur_comp_info[ci];
                start_col = MCU_col_num * comp_ptr->MCU_width;
                for (yindex = 0; yindex < comp_ptr->MCU_height; yindex++) {
                    buffer_ptr = buffer[ci][yindex+yoffset] + start_col;
                    for (xindex = 0; xindex < comp_ptr->MCU_width; xindex++) {
                        coef->MCU_buffer[blkcn++] = buffer_ptr++;
                    }
                }
            }
            /* Try to write the MCU. */
            if (!(*cinfo->entropy->encode_mcu) (cinfo, coef->MCU_buffer)) {
                /* Suspension forced; update state counters and exit */
                coef->MCU_vert_offset = yoffset;
                coef->mcu_ctr = MCU_col_num;
                return FALSE;
            }
        }
        /* Completed an MCU row, but perhaps not an iMCU row */
        coef->mcu_ctr = 0;
    }
    /* Completed the iMCU row, advance counters for next one */
    coef->iMCU_row_num++;
    start_iMCU_row(cinfo);
    return TRUE;
}

#endif /* FULL_COEF_BUFFER_SUPPORTED */

/*

```

```

* Initialize coefficient buffer controller.
*/

GLOBAL(void)
jinit_c_coef_controller (j_compress_ptr cinfo, boolean need_full_buffer)
{
    my_coef_ptr coef;

    coef = (my_coef_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            sizeof(my_coef_controller));
    cinfo->coef = (struct jpeg_c_coef_controller *) coef;
    coef->pub.start_pass = start_pass_coef;

    /* Create the coefficient buffer. */
    if (need_full_buffer) {
#ifdef FULL_COEF_BUFFER_SUPPORTED
        /* Allocate a full-image virtual array for each component, */
        /* padded to a multiple of samp_factor DCT blocks in each direction. */
        int ci;
        jpeg_component_info *comp_ptr;

        for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
            ci++, comp_ptr++) {
            coef->whole_image[ci] = (*cinfo->mem->request_virt_barray)
                ((j_common_ptr) cinfo, JPOOL_IMAGE, FALSE,
                (JDIMENSION) jround_up((long) comp_ptr->width_in_blocks,
                    (long) comp_ptr->h_samp_factor),
                (JDIMENSION) jround_up((long) comp_ptr->height_in_blocks,
                    (long) comp_ptr->v_samp_factor),
                (JDIMENSION) comp_ptr->v_samp_factor);
        }
    }
    #else
        ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
    #endif
    else {
        /* We only need a single-MCU buffer. */
        JBLOCKROW buffer;
        int i;

        buffer = (JBLOCKROW)
            (*cinfo->mem->alloc_large) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                C_MAX_BLOCKS_IN_MCU * sizeof(JBLOCK));
        for (i = 0; i < C_MAX_BLOCKS_IN_MCU; i++) {
            coef->MCU_buffer[i] = buffer + i;
        }
        coef->whole_image[0] = NULL; /* flag for no virtual arrays */
    }
}

```

```

/*
 * jccolor.c
 *
 * Copyright (C) 1991-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains input colorspace conversion routines.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/* Private subobject */

typedef struct {
  struct jpeg_color_converter pub; /* public fields */

  /* Private state for RGB->YCC conversion */
  INT32 * rgb_ycc_tab; /* => table for RGB to YCbCr conversion */
} my_color_converter;

typedef my_color_converter * my_cconvert_ptr;

/***** RGB -> YCbCr conversion: most common case *****/

/*
 * YCbCr is defined per CCIR 601-1, except that Cb and Cr are
 * normalized to the range 0..MAXJSAMPLE rather than -0.5 .. 0.5.
 * The conversion equations to be implemented are therefore
 * Y = 0.29900 * R + 0.58700 * G + 0.11400 * B
 * Cb = -0.16874 * R - 0.33126 * G + 0.50000 * B + CENTERJSAMPLE
 * Cr = 0.50000 * R - 0.41869 * G - 0.08131 * B + CENTERJSAMPLE
 * (These numbers are derived from TIFF 6.0 section 21, dated 3-June-92.)
 * Note: older versions of the IJG code used a zero offset of MAXJSAMPLE/2,
 * rather than CENTERJSAMPLE, for Cb and Cr. This gave equal positive and
 * negative swings for Cb/Cr, but meant that grayscale values (Cb=Cr=0)
 * were not represented exactly. Now we sacrifice exact representation of
 * maximum red and maximum blue in order to get exact grayscales.
 *
 * To avoid floating-point arithmetic, we represent the fractional constants
 * as integers scaled up by 2^16 (about 4 digits precision); we have to divide
 * the products by 2^16, with appropriate rounding, to get the correct answer.
 *
 * For even more speed, we avoid doing any multiplications in the inner loop
 * by precalculating the constants times R,G,B for all possible values.
 * For 8-bit JSAMPLEs this is very reasonable (only 256 entries per table);
 * for 12-bit samples it is still acceptable. It's not very reasonable for
 * 16-bit samples, but if you want lossless storage you shouldn't be changing
 * colorspace anyway.
 * The CENTERJSAMPLE offsets and the rounding fudge-factor of 0.5 are included
 * in the tables to save adding them separately in the inner loop.
 */

#define SCALEBITS 16 /* speediest right-shift on some machines */
#define CBCR_OFFSET ((INT32) CENTERJSAMPLE << SCALEBITS)
#define ONE_HALF ((INT32) 1 << (SCALEBITS-1))
#define FIX(x) ((INT32) ((x) * (1L<<SCALEBITS) + 0.5))

/* We allocate one big table and divide it up into eight parts, instead of
 * doing eight alloc_small requests. This lets us use a single table base
 * address, which can be held in a register in the inner loops on many
 * machines (more than can hold all eight addresses, anyway).
 */

#define R_Y_OFF 0 /* offset to R => Y section */
#define G_Y_OFF (1*(MAXJSAMPLE+1)) /* offset to G => Y section */
#define B_Y_OFF (2*(MAXJSAMPLE+1)) /* etc. */
#define R_CB_OFF (3*(MAXJSAMPLE+1))
#define G_CB_OFF (4*(MAXJSAMPLE+1))
#define B_CB_OFF (5*(MAXJSAMPLE+1))
#define R_CR_OFF B_CB_OFF /* B=>Cb, R=>Cr are the same */
#define G_CR_OFF (6*(MAXJSAMPLE+1))
#define B_CR_OFF (7*(MAXJSAMPLE+1))
#define TABLE_SIZE (8*(MAXJSAMPLE+1))

```

```

/*
 * Initialize for RGB->YCC colorspace conversion.
 */

METHODDEF(void)
rgb_ycc_start (j_compress_ptr cinfo)
{
    my_cconvert_ptr cconvert = (my_cconvert_ptr) cinfo->cconvert;
    INT32 * rgb_ycc_tab;
    INT32 i;

    /* Allocate and fill in the conversion tables. */
    cconvert->rgb_ycc_tab = rgb_ycc_tab = (INT32 *)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            (TABLE_SIZE * SIZEOF(INT32)));

    for (i = 0; i <= MAXJSAMPLE; i++) {
        rgb_ycc_tab[i+R_Y_OFF] = FIX(0.29900) * i;
        rgb_ycc_tab[i+G_Y_OFF] = FIX(0.58700) * i;
        rgb_ycc_tab[i+B_Y_OFF] = FIX(0.11400) * i      + ONE_HALF;
        rgb_ycc_tab[i+R_CB_OFF] = (-FIX(0.16874)) * i;
        rgb_ycc_tab[i+G_CB_OFF] = (-FIX(0.33126)) * i;
        /* We use a rounding fudge-factor of 0.5-epsilon for Cb and Cr.
         * This ensures that the maximum output will round to MAXJSAMPLE
         * not MAXJSAMPLE+1, and thus that we don't have to range-limit.
         */
        rgb_ycc_tab[i+B_CB_OFF] = FIX(0.50000) * i      + CBCR_OFFSET + ONE_HALF-1;
    /* B=>Cb and R=>Cr tables are the same
        rgb_ycc_tab[i+R_CR_OFF] = FIX(0.50000) * i      + CBCR_OFFSET + ONE_HALF-1;
    */
        rgb_ycc_tab[i+G_CR_OFF] = (-FIX(0.41869)) * i;
        rgb_ycc_tab[i+B_CR_OFF] = (-FIX(0.08131)) * i;
    }
}

/*
 * Convert some rows of samples to the JPEG colorspace.
 *
 * Note that we change from the application's interleaved-pixel format
 * to our internal noninterleaved, one-plane-per-component format.
 * The input buffer is therefore three times as wide as the output buffer.
 *
 * A starting row offset is provided only for the output buffer. The caller
 * can easily adjust the passed input_buf value to accommodate any row
 * offset required on that side.
 */

METHODDEF(void)
rgb_ycc_convert (j_compress_ptr cinfo,
    JSAMPARRAY input_buf, JSAMPIMAGE output_buf,
    JDIMENSION output_row, int num_rows)
{
    my_cconvert_ptr cconvert = (my_cconvert_ptr) cinfo->cconvert;
    register int r, g, b;
    register INT32 * ctab = cconvert->rgb_ycc_tab;
    register JSAMPROW inptr;
    register JSAMPROW outptr0, outptr1, outptr2;
    register JDIMENSION col;
    JDIMENSION num_cols = cinfo->image_width;

    while (--num_rows >= 0) {
        inptr = *input_buf++;
        outptr0 = output_buf[0][output_row];
        outptr1 = output_buf[1][output_row];
        outptr2 = output_buf[2][output_row];
        output_row++;
        for (col = 0; col < num_cols; col++) {
            r = GETJSAMPLE(inptr[RGB_RED]);
            g = GETJSAMPLE(inptr[RGB_GREEN]);
            b = GETJSAMPLE(inptr[RGB_BLUE]);
            inptr += RGB_PIXELSIZE;
            /* If the inputs are 0..MAXJSAMPLE, the outputs of these equations
             * must be too; we do not need an explicit range-limiting operation.
             * Hence the value being shifted is never negative, and we don't
             * need the general RIGHT_SHIFT macro.
             */
            /* Y */
            outptr0[col] = (JSAMPLE)
                ((ctab[r+R_Y_OFF] + ctab[g+G_Y_OFF] + ctab[b+B_Y_OFF])

```

```

        >> SCALEBITS);
/* Cb */
outptr1[col] = (JSAMPLE)
    ((ctab[r+R_CB_OFF] + ctab[g+G_CB_OFF] + ctab[b+B_CB_OFF])
    >> SCALEBITS);
/* Cr */
outptr2[col] = (JSAMPLE)
    ((ctab[r+R_CR_OFF] + ctab[g+G_CR_OFF] + ctab[b+B_CR_OFF])
    >> SCALEBITS);
    }
}

/***** Cases other than RGB -> YCbCr *****/

/*
 * Convert some rows of samples to the JPEG colorspace.
 * This version handles RGB->grayscale conversion, which is the same
 * as the RGB->Y portion of RGB->YCbCr.
 * We assume rgb_ycc_start has been called (we only use the Y tables).
 */

METHODDEF(void)
rgb_gray_convert (j_compress_ptr cinfo,
                  JSAMPARRAY input_buf, JSAMPIMAGE output_buf,
                  JDIMENSION output_row, int num_rows)
{
    my_cconvert_ptr cconvert = (my_cconvert_ptr) cinfo->cconvert;
    register int r, g, b;
    register INT32 * ctab = cconvert->rgb_ycc_tab;
    register JSAMPROW inptr;
    register JSAMPROW outptr;
    register JDIMENSION col;
    JDIMENSION num_cols = cinfo->image_width;

    while (--num_rows >= 0) {
        inptr = *input_buf++;
        outptr = output_buf[0][output_row];
        output_row++;
        for (col = 0; col < num_cols; col++) {
            r = GETJSAMPLE(inptr[RGB_RED]);
            g = GETJSAMPLE(inptr[RGB_GREEN]);
            b = GETJSAMPLE(inptr[RGB_BLUE]);
            inptr += RGB_PIXELSIZE;
            /* Y */
            outptr[col] = (JSAMPLE)
                ((ctab[r+R_Y_OFF] + ctab[g+G_Y_OFF] + ctab[b+B_Y_OFF])
                >> SCALEBITS);
        }
    }

/*
 * Convert some rows of samples to the JPEG colorspace.
 * This version handles Adobe-style CMYK->YCK conversion,
 * where we convert R=1-C, G=1-M, and B=1-Y to YCbCr using the same
 * conversion as above, while passing K (black) unchanged.
 * We assume rgb_ycc_start has been called.
 */

METHODDEF(void)
cmyk_ycck_convert (j_compress_ptr cinfo,
                   JSAMPARRAY input_buf, JSAMPIMAGE output_buf,
                   JDIMENSION output_row, int num_rows)
{
    my_cconvert_ptr cconvert = (my_cconvert_ptr) cinfo->cconvert;
    register int r, g, b;
    register INT32 * ctab = cconvert->rgb_ycc_tab;
    register JSAMPROW inptr;
    register JSAMPROW outptr0, outptr1, outptr2, outptr3;
    register JDIMENSION col;
    JDIMENSION num_cols = cinfo->image_width;

    while (--num_rows >= 0) {
        inptr = *input_buf++;
        outptr0 = output_buf[0][output_row];
        outptr1 = output_buf[1][output_row];

```

```

outptr2 = output_buf[2][output_row];
outptr3 = output_buf[3][output_row];
output_row++;
for (col = 0; col < num_cols; col++) {
    r = MAXJSAMPLE - GETJSAMPLE(inptr[0]);
    g = MAXJSAMPLE - GETJSAMPLE(inptr[1]);
    b = MAXJSAMPLE - GETJSAMPLE(inptr[2]);
    /* K passes through as-is */
    outptr3[col] = inptr[3]; /* don't need GETJSAMPLE here */
    inptr += 4;
    /* If the inputs are 0..MAXJSAMPLE, the outputs of these equations
     * must be too; we do not need an explicit range-limiting operation.
     * Hence the value being shifted is never negative, and we don't
     * need the general RIGHT_SHIFT macro.
     */
    /* Y */
    outptr0[col] = (JSAMPLE)
        ((ctab[r+R_Y_OFF] + ctab[g+G_Y_OFF] + ctab[b+B_Y_OFF])
         >> SCALEBITS);
    /* Cb */
    outptr1[col] = (JSAMPLE)
        ((ctab[r+R_CB_OFF] + ctab[g+G_CB_OFF] + ctab[b+B_CB_OFF])
         >> SCALEBITS);
    /* Cr */
    outptr2[col] = (JSAMPLE)
        ((ctab[r+R_CR_OFF] + ctab[g+G_CR_OFF] + ctab[b+B_CR_OFF])
         >> SCALEBITS);
}
}
}

```

```

/*
 * Convert some rows of samples to the JPEG colorspace.
 * This version handles grayscale output with no conversion.
 * The source can be either plain grayscale or YCbCr (since Y == gray).
 */

```

```

METHODDEF(void)
grayscale_convert (j_compress_ptr cinfo,
                  JSAMPARRAY input_buf, JSAMPIMAGE output_buf,
                  JDIMENSION output_row, int num_rows)
{
    register JSAMPROW inptr;
    register JSAMPROW outptr;
    register JDIMENSION col;
    JDIMENSION num_cols = cinfo->image_width;
    int instride = cinfo->input_components;

    while (--num_rows >= 0) {
        inptr = *input_buf++;
        outptr = output_buf[0][output_row];
        output_row++;
        for (col = 0; col < num_cols; col++) {
            outptr[col] = inptr[0]; /* don't need GETJSAMPLE() here */
            inptr += instride;
        }
    }
}

```

```

/*
 * Convert some rows of samples to the JPEG colorspace.
 * This version handles multi-component colorspace without conversion.
 * We assume input_components == num_components.
 */

```

```

METHODDEF(void)
null_convert (j_compress_ptr cinfo,
              JSAMPARRAY input_buf, JSAMPIMAGE output_buf,
              JDIMENSION output_row, int num_rows)
{
    register JSAMPROW inptr;
    register JSAMPROW outptr;
    register JDIMENSION col;
    register int ci;
    int nc = cinfo->num_components;
    JDIMENSION num_cols = cinfo->image_width;

    while (--num_rows >= 0) {

```



```

/* It seems fastest to make a separate pass for each component. */
for (ci = 0; ci < nc; ci++) {
    inptr = *input_buf;
    outptr = output_buf[ci][output_row];
    for (col = 0; col < num_cols; col++) {
        outptr[col] = inptr[ci]; /* don't need GETJSAMPLE() here */
        inptr += nc;
    }
    input_buf++;
    output_row++;
}

/*
 * Empty method for start_pass.
 */

METHODDEF(void)
null_method (j_compress_ptr cinfo)
{
    /* no work needed */
}

/*
 * Module initialization routine for input colorspace conversion.
 */

GLOBAL(void)
jinit_color_converter (j_compress_ptr cinfo)
{
    my_cconvert_ptr cconvert;
    cconvert = (my_cconvert_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
        sizeof(my_color_converter));
    cinfo->cconvert = (struct jpeg_color_converter *) cconvert;
    /* set start_pass to null method until we find out differently */
    cconvert->pub.start_pass = null_method;

    /* Make sure input_components agrees with in_color_space */
    switch (cinfo->in_color_space) {
        case JCS_GRAYSCALE:
            if (cinfo->input_components != 1)
                ERREXIT(cinfo, JERR_BAD_IN_COLORSPACE);
            break;
        case JCS_RGB:
            #if RGB_PIXELSIZE != 3
            if (cinfo->input_components != RGB_PIXELSIZE)
                ERREXIT(cinfo, JERR_BAD_IN_COLORSPACE);
            break;
            #endif /* else share code with YCbCr */
        case JCS_YCbCr:
            if (cinfo->input_components != 3)
                ERREXIT(cinfo, JERR_BAD_IN_COLORSPACE);
            break;
        case JCS_CMYK:
        case JCS_YCCK:
            if (cinfo->input_components != 4)
                ERREXIT(cinfo, JERR_BAD_IN_COLORSPACE);
            break;
        default:
            /* JCS_UNKNOWN can be anything */
            if (cinfo->input_components < 1)
                ERREXIT(cinfo, JERR_BAD_IN_COLORSPACE);
            break;
    }

    /* Check num_components, set conversion method based on requested space */
    switch (cinfo->jpeg_color_space) {
        case JCS_GRAYSCALE:
            if (cinfo->num_components != 1)
                ERREXIT(cinfo, JERR_BAD_J_COLORSPACE);
            if (cinfo->in_color_space == JCS_GRAYSCALE)
                cconvert->pub.color_convert = grayscale_convert;
    }

```

```

else if (cinfo->in_color_space == JCS_RGB) {
    cconvert->pub.start_pass = rgb_ycc_start;
    cconvert->pub.color_convert = rgb_gray_convert;
} else if (cinfo->in_color_space == JCS_YCbCr)
    cconvert->pub.color_convert = grayscale_convert;
else
    ERREXIT(cinfo, JERR_CONVERSION_NOTIMPL);
break;

case JCS_RGB:
    if (cinfo->num_components != 3)
        ERREXIT(cinfo, JERR_BAD_J_COLORSPACE);
    if (cinfo->in_color_space == JCS_RGB && RGB_PIXELSIZE == 3)
        cconvert->pub.color_convert = null_convert;
    else
        ERREXIT(cinfo, JERR_CONVERSION_NOTIMPL);
    break;

case JCS_YCbCr:
    if (cinfo->num_components != 3)
        ERREXIT(cinfo, JERR_BAD_J_COLORSPACE);
    if (cinfo->in_color_space == JCS_RGB) {
        cconvert->pub.start_pass = rgb_ycc_start;
        cconvert->pub.color_convert = rgb_ycc_convert;
    } else if (cinfo->in_color_space == JCS_YCbCr)
        cconvert->pub.color_convert = null_convert;
    else
        ERREXIT(cinfo, JERR_CONVERSION_NOTIMPL);
    break;

case JCS_CMYK:
    if (cinfo->num_components != 4)
        ERREXIT(cinfo, JERR_BAD_J_COLORSPACE);
    if (cinfo->in_color_space == JCS_CMYK)
        cconvert->pub.color_convert = null_convert;
    else
        ERREXIT(cinfo, JERR_CONVERSION_NOTIMPL);
    break;

case JCS_YCCK:
    if (cinfo->num_components != 4)
        ERREXIT(cinfo, JERR_BAD_J_COLORSPACE);
    if (cinfo->in_color_space == JCS_CMYK) {
        cconvert->pub.start_pass = rgb_ycc_start;
        cconvert->pub.color_convert = cmyk_ycck_convert;
    } else if (cinfo->in_color_space == JCS_YCCK)
        cconvert->pub.color_convert = null_convert;
    else
        ERREXIT(cinfo, JERR_CONVERSION_NOTIMPL);
    break;

default:
    /* allow null conversion of JCS_UNKNOWN */
    if (cinfo->jpeg_color_space != cinfo->in_color_space ||
        cinfo->num_components != cinfo->input_components)
        ERREXIT(cinfo, JERR_CONVERSION_NOTIMPL);
    cconvert->pub.color_convert = null_convert;
    break;
}
}

```

```

/*
 * jcdctmgr.c
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains the forward-DCT management logic.
 * This code selects a particular DCT implementation to be used,
 * and it performs related housekeeping chores including coefficient
 * quantization.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jdct.h"          /* Private declarations for DCT subsystem */

/* Private subobject for this module */

typedef struct {
  struct jpeg_forward_dct pub; /* public fields */

  /* Pointer to the DCT routine actually in use */
  forward_DCT_method_ptr do_dct;

  /* The actual post-DCT divisors --- not identical to the quant table
   * entries, because of scaling (especially for an unnormalized DCT).
   * Each table is given in normal array order.
   */
  DCTELEM * divisors[NUM_QUANT_TBLS];

#ifdef DCT_FLOAT_SUPPORTED
  /* Same as above for the floating-point case. */
  float_DCT_method_ptr do_float_dct;
  FAST_FLOAT * float_divisors[NUM_QUANT_TBLS];
#endif
} my_fdct_controller;

typedef my_fdct_controller * my_fdct_ptr;

/*
 * Initialize for a processing pass.
 * Verify that all referenced Q-tables are present, and set up
 * the divisor table for each one.
 * In the current implementation, DCT of all components is done during
 * the first pass, even if only some components will be output in the
 * first scan. Hence all components should be examined here.
 */

METHODDEF(void)
start_pass_fdctmgr (j_compress_ptr cinfo)
{
  my_fdct_ptr fdct = (my_fdct_ptr) cinfo->fdct;
  int ci, qtblno, i;
  jpeg_component_info *comp_ptr;
  JQUANT_TBL * qtbl;
  DCTELEM * dtbl;

  for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
       ci++, comp_ptr++) {
    qtblno = comp_ptr->quant_tbl_no;
    /* Make sure specified quantization table is present */
    if (qtblno < 0 || qtblno >= NUM_QUANT_TBLS ||
        cinfo->quant_tbl_ptrs[qtblno] == NULL)
      ERREXIT1(cinfo, JERR_NO_QUANT_TABLE, qtblno);
    qtbl = cinfo->quant_tbl_ptrs[qtblno];
    /* Compute divisors for this quant table */
    /* We may do this more than once for same table, but it's not a big deal */
    switch (cinfo->dct_method) {
#ifdef DCT_ISLOW_SUPPORTED
      case JDCT_ISLOW:
        /* For LL&M IDCT method, divisors are equal to raw quantization
         * coefficients multiplied by 8 (to counteract scaling).
         */
        if (fdct->divisors[qtblno] == NULL) {
          fdct->divisors[qtblno] = (DCTELEM *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,

```

```

        DCTSIZE2 * sizeof(DCTELEM));
    }
    dtbl = fdct->divisors[qtblno];
    for (i = 0; i < DCTSIZE2; i++) {
        dtbl[i] = ((DCTELEM) qtbl->quantval[i]) << 3;
    }
    break;
#endif
#ifdef DCT_IFAST_SUPPORTED
case JDCT_IFAST:
{
    /* For AA&N IDCT method, divisors are equal to quantization
     * coefficients scaled by scalefactor[row]*scalefactor[col], where
     *   scalefactor[0] = 1
     *   scalefactor[k] = cos(k*PI/16) * sqrt(2)    for k=1..7
     * We apply a further scale factor of 8.
     */
#define CONST_BITS 14
    static const INT16 aanscales[DCTSIZE2] = {
        /* precomputed values scaled up by 14 bits */
        16384, 22725, 21407, 19266, 16384, 12873, 8867, 4520,
        22725, 31521, 29692, 26722, 22725, 17855, 12299, 6270,
        21407, 29692, 27969, 25172, 21407, 16819, 11585, 5906,
        19266, 26722, 25172, 22654, 19266, 15137, 10426, 5315,
        16384, 22725, 21407, 19266, 16384, 12873, 8867, 4520,
        12873, 17855, 16819, 15137, 12873, 10114, 6967, 3552,
        8867, 12299, 11585, 10426, 8867, 6967, 4799, 2446,
        4520, 6270, 5906, 5315, 4520, 3552, 2446, 1247
    };
    SHIFT_TEMPS

    if (fdct->divisors[qtblno] == NULL) {
        fdct->divisors[qtblno] = (DCTELEM *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                DCTSIZE2 * sizeof(DCTELEM));
    }
    dtbl = fdct->divisors[qtblno];
    for (i = 0; i < DCTSIZE2; i++) {
        dtbl[i] = (DCTELEM)
            DESCALE(MULTIPLY16V16((INT32) qtbl->quantval[i],
                (INT32) aanscales[i]),
                CONST_BITS-3);
    }
    break;
#endif
#ifdef DCT_FLOAT_SUPPORTED
case JDCT_FLOAT:
{
    /* For float AA&N IDCT method, divisors are equal to quantization
     * coefficients scaled by scalefactor[row]*scalefactor[col], where
     *   scalefactor[0] = 1
     *   scalefactor[k] = cos(k*PI/16) * sqrt(2)    for k=1..7
     * We apply a further scale factor of 8.
     * What's actually stored is 1/divisor so that the inner loop can
     * use a multiplication rather than a division.
     */
    FAST_FLOAT * fdtbl;
    int row, col;
    static const double aanscalefactor[DCTSIZE] = {
        1.0, 1.387039845, 1.306562965, 1.175875602,
        1.0, 0.785694958, 0.541196100, 0.275899379
    };

    if (fdct->float_divisors[qtblno] == NULL) {
        fdct->float_divisors[qtblno] = (FAST_FLOAT *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                DCTSIZE2 * sizeof(FAST_FLOAT));
    }
    fdtbl = fdct->float_divisors[qtblno];
    i = 0;
    for (row = 0; row < DCTSIZE; row++) {
        for (col = 0; col < DCTSIZE; col++) {
            fdtbl[i] = (FAST_FLOAT)
                (1.0 / (((double) qtbl->quantval[i] *
                    aanscalefactor[row] * aanscalefactor[col] * 8.0)));
            i++;
        }
    }
}
}

```

```

        break;
#endif
    default:
        ERREXIT(cinfo, JERR_NOT_COMPILED);
        break;
    }
}

/*
 * Perform forward DCT on one or more blocks of a component.
 *
 * The input samples are taken from the sample_data[] array starting at
 * position start_row/start_col, and moving to the right for any additional
 * blocks. The quantized coefficients are returned in coef_blocks[].
 */

METHODDEF(void)
forward_DCT (j_compress_ptr cinfo, jpeg_component_info * comp_ptr,
             JSAMPARRAY sample_data, JBLOCKROW coef_blocks,
             JDIMENSION start_row, JDIMENSION start_col,
             JDIMENSION num_blocks)
/* This version is used for integer DCT implementations. */
{
    /* This routine is heavily used, so it's worth coding it tightly. */
    my_fdct_ptr fdct = (my_fdct_ptr) cinfo->fdct;
    forward_DCT_method_ptr do_dct = fdct->do_dct;
    DCTELEM * divisors = fdct->divisors[comp_ptr->quant_tbl_no];
    DCTELEM workspace[DCTSIZE2]; /* work area for FDCT subroutine */
    JDIMENSION bi;

    sample_data += start_row; /* fold in the vertical offset once */

    for (bi = 0; bi < num_blocks; bi++, start_col += DCTSIZE) {
        /* Load data into workspace, applying unsigned->signed conversion */
        { register DCTELEM *workspaceptr;
          register JSAMPROW elem_ptr;
          register int elem_r;

          workspaceptr = workspace;
          for (elem_r = 0; elem_r < DCTSIZE; elem_r++) {
              elem_ptr = sample_data[elem_r] + start_col;
              /* unroll the inner loop */
              *workspaceptr++ = GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE;
              *workspaceptr++ = GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE;
              *workspaceptr++ = GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE;
              *workspaceptr++ = GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE;
              *workspaceptr++ = GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE;
              *workspaceptr++ = GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE;
              *workspaceptr++ = GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE;
              *workspaceptr++ = GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE;
          }
        }
        /* Perform the DCT */
        (*do_dct) (workspace);

        /* Quantize/descale the coefficients, and store into coef_blocks[] */
        { register DCTELEM temp, qval;
          register int i;
          register JCOEFPTR output_ptr = coef_blocks[bi];

          for (i = 0; i < DCTSIZE2; i++) {
              qval = divisors[i];
              temp = workspace[i];
              /* Divide the coefficient value by qval, ensuring proper rounding.
               * Since C does not specify the direction of rounding for negative
               * quotients, we have to force the dividend positive for portability.
               *
               * In most files, at least half of the output values will be zero
               * (at default quantization settings, more like three-quarters...)
               * so we should ensure that this case is fast. On many machines,

```

```

* a comparison is enough cheaper than a divide to make a special test
* a win. Since both inputs will be nonnegative, we need only test
* for a < b to discover whether a/b is 0.
* If your machine's division is fast enough, define FAST_DIVIDE.
*/
#ifdef FAST_DIVIDE
#define DIVIDE_BY(a,b)  a /= b
#else
#define DIVIDE_BY(a,b)  if (a >= b) a /= b; else a = 0
#endif
    if (temp < 0) {
        temp = -temp;
        temp += qval>>1; /* for rounding */
        DIVIDE_BY(temp, qval);
        temp = -temp;
    } else {
        temp += qval>>1; /* for rounding */
        DIVIDE_BY(temp, qval);
    }
    output_ptr[i] = (JCOEF) temp;
}
}

#ifdef DCT_FLOAT_SUPPORTED

METHODDEF(void)
forward_DCT_float (j_compress_ptr cinfo, jpeg_component_info * comp_ptr,
    JSAMPARRAY sample_data, JBLOCKROW coef_blocks,
    JDIMENSION start_row, JDIMENSION start_col,
    JDIMENSION num_blocks)
/* This version is used for floating-point DCT implementations. */
{
    /* This routine is heavily used, so it's worth coding it tightly. */
    my_fdct_ptr fdct = (my_fdct_ptr) cinfo->fdct;
    float_DCT_method_ptr do_dct = fdct->do_float_dct;
    FAST_FLOAT * divisors = fdct->float_divisors[comp_ptr->quant_tbl_no];
    FAST_FLOAT workspace[DCTSIZE2]; /* work area for FDCT subroutine */
    JDIMENSION bi;

    sample_data += start_row; /* fold in the vertical offset once */

    for (bi = 0; bi < num_blocks; bi++, start_col += DCTSIZE) {
        /* Load data into workspace, applying unsigned->signed conversion */
        { register FAST_FLOAT *workspace_ptr;
          register JSAMPROW elem_ptr;
          register int elem_r;

          workspace_ptr = workspace;
          for (elem_r = 0; elem_r < DCTSIZE; elem_r++) {
              elem_ptr = sample_data[elem_r] + start_col;
              if (DCTSIZE == 8) /* unroll the inner loop */
                  *workspace_ptr++ = (FAST_FLOAT) (GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE);
                  *workspace_ptr++ = (FAST_FLOAT) (GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE);
                  *workspace_ptr++ = (FAST_FLOAT) (GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE);
                  *workspace_ptr++ = (FAST_FLOAT) (GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE);
                  *workspace_ptr++ = (FAST_FLOAT) (GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE);
                  *workspace_ptr++ = (FAST_FLOAT) (GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE);
                  *workspace_ptr++ = (FAST_FLOAT) (GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE);
                  *workspace_ptr++ = (FAST_FLOAT) (GETJSAMPLE(*elem_ptr++) - CENTERJSAMPLE);
              }
          }
        }

        /* Perform the DCT */
        (*do_dct) (workspace);

        /* Quantize/descale the coefficients, and store into coef_blocks[] */
        { register FAST_FLOAT temp;
          register int i;
          register JCOEFPTR output_ptr = coef_blocks[bi];

```

```

    for (i = 0; i < DCTSIZE2; i++) {
/* Apply the quantization and scaling factor */
temp = workspace[i] * divisors[i];
/* Round to nearest integer.
 * Since C does not specify the direction of rounding for negative
 * quotients, we have to force the dividend positive for portability.
 * The maximum coefficient size is +-16K (for 12-bit data), so this
 * code should work for either 16-bit or 32-bit ints.
 */
output_ptr[i] = (JCOEF) ((int) (temp + (FAST_FLOAT) 16384.5) - 16384);
    }
}
}

#endif /* DCT_FLOAT_SUPPORTED */

/*
 * Initialize FDCT manager.
 */

GLOBAL(void)
jinit_forward_dct (j_compress_ptr cinfo)
{
    my_fdct_ptr fdct;
    int i;

    fdct = (my_fdct_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                                   sizeof(my_fdct_controller));
    cinfo->fdct = (struct jpeg_forward_dct *) fdct;
    fdct->pub.start_pass = start_pass_fdctmgr;

    switch (cinfo->dct_method) {
#ifdef DCT_ISLOW_SUPPORTED
    case JDCT_ISLOW:
        fdct->pub.forward_DCT = forward_DCT;
        fdct->do_dct = jpeg_fdct_islow;
        break;
#endif
#ifdef DCT_IFAST_SUPPORTED
    case JDCT_IFAST:
        fdct->pub.forward_DCT = forward_DCT;
        fdct->do_dct = jpeg_fdct_ifast;
        break;
#endif
#ifdef DCT_FLOAT_SUPPORTED
    case JDCT_FLOAT:
        fdct->pub.forward_DCT = forward_DCT_float;
        fdct->do_float_dct = jpeg_fdct_float;
        break;
#endif
    default:
        ERREXIT(cinfo, JERR_NOT_COMPILED);
        break;
    }

    /* Mark divisor tables unallocated */
    for (i = 0; i < NUM_QUANT_TBLS; i++) {
        fdct->divisors[i] = NULL;
#ifdef DCT_FLOAT_SUPPORTED
        fdct->float_divisors[i] = NULL;
#endif
    }
}

```

```

/*
 * jchuff.c
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains Huffman entropy encoding routines.
 *
 * Much of the complexity here has to do with supporting output suspension.
 * If the data destination module demands suspension, we want to be able to
 * back up to the start of the current MCU. To do this, we copy state
 * variables into local working storage, and update them back to the
 * permanent JPEG objects only upon successful completion of an MCU.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jchuff.h" /* Declarations shared with jcphuff.c */

/* Expanded entropy encoder object for Huffman encoding.
 *
 * The savable_state subrecord contains fields that change within an MCU,
 * but must not be updated permanently until we complete the MCU.
 */

typedef struct {
  INT32 put_buffer; /* current bit-accumulation buffer */
  int put_bits; /* # of bits now in it */
  int last_dc_val[MAX_COMPS_IN_SCAN]; /* last DC coef for each component */
} savable_state;

/* This macro is to work around compilers with missing or broken
 * structure assignment. You'll need to fix this code if you have
 * such a compiler and you change MAX_COMPS_IN_SCAN.
 */

#ifdef NO_STRUCT_ASSIGN
#define ASSIGN_STATE(dest,src) ((dest) = (src))
#else
#define ASSIGN_STATE(dest,src) \
  ((dest).put_buffer = (src).put_buffer, \
   (dest).put_bits = (src).put_bits, \
   (dest).last_dc_val[0] = (src).last_dc_val[0], \
   (dest).last_dc_val[1] = (src).last_dc_val[1], \
   (dest).last_dc_val[2] = (src).last_dc_val[2], \
   (dest).last_dc_val[3] = (src).last_dc_val[3])
#endif

typedef struct {
  struct jpeg_entropy_encoder pub; /* public fields */

  savable_state saved; /* Bit buffer & DC state at start of MCU */

  /* These fields are NOT loaded into local working state. */
  unsigned int restarts_to_go; /* MCUs left in this restart interval */
  int next_restart_num; /* next restart number to write (0-7) */

  /* Pointers to derived tables (these workspaces have image lifespan) */
  c_derived_tbl * dc_derived_tbls[NUM_HUFF_TBLS];
  c_derived_tbl * ac_derived_tbls[NUM_HUFF_TBLS];

#ifdef ENTROPY_OPT_SUPPORTED /* Statistics tables for optimization */
  long * dc_count_ptrs[NUM_HUFF_TBLS];
  long * ac_count_ptrs[NUM_HUFF_TBLS];
#endif
} huff_entropy_encoder;

typedef huff_entropy_encoder * huff_entropy_ptr;

/* Working state while writing an MCU.
 * This struct contains all the fields that are needed by subroutines.
 */

typedef struct {

```



```

JOCTET * next_output_byte; /* => next byte to write in buffer */
size_t free_in_buffer; /* # of byte spaces remaining in buffer */
savable_state cur; /* Current bit buffer & DC state */
j_compress_ptr cinfo; /* dump_buffer needs access to this */
} working_state;

/* Forward declarations */
METHODDEF(boolean) encode_mcu_huff JPP((j_compress_ptr cinfo,
JBLOCKROW *MCU_data));
METHODDEF(void) finish_pass_huff JPP((j_compress_ptr cinfo));
#ifdef ENTROPY_OPT_SUPPORTED
METHODDEF(boolean) encode_mcu_gather JPP((j_compress_ptr cinfo,
JBLOCKROW *MCU_data));
METHODDEF(void) finish_pass_gather JPP((j_compress_ptr cinfo));
#endif

/*
 * Initialize for a Huffman-compressed scan.
 * If gather_statistics is TRUE, we do not output anything during the scan,
 * just count the Huffman symbols used and generate Huffman code tables.
 */

METHODDEF(void)
start_pass_huff (j_compress_ptr cinfo, boolean gather_statistics)
{
    huff_entropy_ptr entropy = (huff_entropy_ptr) cinfo->entropy;
    int ci, dctbl, actbl;
    jpeg_component_info * comp_ptr;

    if (gather_statistics) {
#ifdef ENTROPY_OPT_SUPPORTED
        entropy->pub.encode_mcu = encode_mcu_gather;
        entropy->pub.finish_pass = finish_pass_gather;
    #else
        ERREXIT(cinfo, JERR_NOT_COMPILED);
    #endif
    } else {
        entropy->pub.encode_mcu = encode_mcu_huff;
        entropy->pub.finish_pass = finish_pass_huff;
    }

    for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
        comp_ptr = cinfo->cur_comp_info[ci];
        dctbl = comp_ptr->dc_tbl_no;
        actbl = comp_ptr->ac_tbl_no;
        if (gather_statistics) {
#ifdef ENTROPY_OPT_SUPPORTED
            /* Check for invalid table indexes */
            /* (make_c_derived_tbl does this in the other path) */
            if (dctbl < 0 || dctbl >= NUM_HUFF_TBLS)
                ERREXIT1(cinfo, JERR_NO_HUFF_TABLE, dctbl);
            if (actbl < 0 || actbl >= NUM_HUFF_TBLS)
                ERREXIT1(cinfo, JERR_NO_HUFF_TABLE, actbl);
            /* Allocate and zero the statistics tables */
            /* Note that jpeg_gen_optimal_table expects 257 entries in each table! */
            if (entropy->dc_count_ptrs[dctbl] == NULL)
                entropy->dc_count_ptrs[dctbl] = (long *)
                    (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                    257 * sizeof(long));
            MEMZERO(entropy->dc_count_ptrs[dctbl], 257 * sizeof(long));
            if (entropy->ac_count_ptrs[actbl] == NULL)
                entropy->ac_count_ptrs[actbl] = (long *)
                    (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                    257 * sizeof(long));
            MEMZERO(entropy->ac_count_ptrs[actbl], 257 * sizeof(long));
        #endif
        } else {
            /* Compute derived values for Huffman tables */
            /* We may do this more than once for a table, but it's not expensive */
            jpeg_make_c_derived_tbl(cinfo, TRUE, dctbl,
                & entropy->dc_derived_tbls[dctbl]);
            jpeg_make_c_derived_tbl(cinfo, FALSE, actbl,
                & entropy->ac_derived_tbls[actbl]);
        }
        /* Initialize DC predictions to 0 */
        entropy->saved.last_dc_val[ci] = 0;
    }
}

```

```

/* Initialize bit buffer to empty */
entropy->saved.put_buffer = 0;
entropy->saved.put_bits = 0;

/* Initialize restart stuff */
entropy->restarts_to_go = cinfo->restart_interval;
entropy->next_restart_num = 0;
}

/*
 * Compute the derived values for a Huffman table.
 * This routine also performs some validation checks on the table.
 *
 * Note this is also used by jcphuff.c.
 */

GLOBAL(void)
jpeg_make_c_derived_tbl (j_compress_ptr cinfo, boolean isDC, int tblno,
                        c_derived_tbl ** pdtbl)
{
    JHUFF_TBL *htbl;
    c_derived_tbl *dtbl;
    int p, i, l, lastp, si, maxsymbol;
    char huffsize[257];
    unsigned int huffcode[257];
    unsigned int code;

    /* Note that huffsize[] and huffcode[] are filled in code-length order,
     * paralleling the order of the symbols themselves in htbl->huffval[].
     */

    /* Find the input Huffman table */
    if (tblno < 0 || tblno >= NUM_HUFF_TBLS)
        ERREXIT1(cinfo, JERR_NO_HUFF_TABLE, tblno);
    htbl =
        isDC ? cinfo->dc_huff_tbl_ptrs[tblno] : cinfo->ac_huff_tbl_ptrs[tblno];
    if (htbl == NULL)
        ERREXIT1(cinfo, JERR_NO_HUFF_TABLE, tblno);

    /* Allocate a workspace if we haven't already done so. */
    if (*pdtbl == NULL)
        *pdtbl = (c_derived_tbl *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                                       sizeof(c_derived_tbl));
    dtbl = *pdtbl;

    /* Figure C.1: make table of Huffman code length for each symbol */
    p = 0;
    for (l = 1; l <= 16; l++) {
        i = (int) htbl->bits[l];
        if (i < 0 || p + i > 256) /* protect against table overrun */
            ERREXIT(cinfo, JERR_BAD_HUFF_TABLE);
        while (i--)
            huffsize[p++] = (char) l;
    }
    huffsize[p] = 0;
    lastp = p;

    /* Figure C.2: generate the codes themselves */
    /* We also validate that the counts represent a legal Huffman code tree. */

    code = 0;
    si = huffsize[0];
    p = 0;
    while (huffsize[p]) {
        while (((int) huffsize[p]) == si) {
            huffcode[p++] = code;
            code++;
        }
        /* code is now 1 more than the last code used for codelength si; but
         * it must still fit in si bits, since no code is allowed to be all ones.
         */
        if (((INT32) code) >= (((INT32) 1) << si))
            ERREXIT(cinfo, JERR_BAD_HUFF_TABLE);
        code <<= 1;
        si++;
    }
}

```

```

/* Figure C.3: generate encoding tables */
/* These are code and size indexed by symbol value */

/* Set all codeless symbols to have code length 0;
 * this lets us detect duplicate VAL entries here, and later
 * allows emit_bits to detect any attempt to emit such symbols.
 */
MEMZERO(dtbl->ehufsi, SIZEOF(dtbl->ehufsi));

/* This is also a convenient place to check for out-of-range
 * and duplicated VAL entries. We allow 0..255 for AC symbols
 * but only 0..15 for DC. (We could constrain them further
 * based on data depth and mode, but this seems enough.)
 */
maxsymbol = isDC ? 15 : 255;

for (p = 0; p < lastp; p++) {
    i = htbl->huffval[p];
    if (i < 0 || i > maxsymbol || dtbl->ehufsi[i])
        ERREXIT(cinfo, JERR_BAD_HUFF_TABLE);
    dtbl->ehufco[i] = huffcode[p];
    dtbl->ehufsi[i] = huffsize[p];
}

/* Outputting bytes to the file */

/* Emit a byte, taking 'action' if must suspend. */
#define emit_byte(state,val,action) \
    { *(state)->next_output_byte++ = (JOCTET) (val); \
      if (--(state)->free_in_buffer == 0) \
          if (! dump_buffer(state)) \
              { action; } }

LOCAL(boolean)
dump_buffer (working_state * state)
/* Empty the output buffer; return TRUE if successful, FALSE if must suspend */
{
    struct jpeg_destination_mgr * dest = state->cinfo->dest;
    if (! (*dest->empty_output_buffer) (state->cinfo))
        return FALSE;
    /* After a successful buffer dump, must reset buffer pointers */
    state->next_output_byte = dest->next_output_byte;
    state->free_in_buffer = dest->free_in_buffer;
    return TRUE;
}

/* Outputting bits to the file */

/* Only the right 24 bits of put_buffer are used; the valid bits are
 * left-justified in this part. At most 16 bits can be passed to emit_bits
 * in one call, and we never retain more than 7 bits in put_buffer
 * between calls, so 24 bits are sufficient.
 */

INLINE
LOCAL(boolean)
emit_bits (working_state * state, unsigned int code, int size)
/* Emit some bits; return TRUE if successful, FALSE if must suspend */
{
    /* This routine is heavily used, so it's worth coding tightly. */
    register INT32 put_buffer = (INT32) code;
    register int put_bits = state->cur.put_bits;

    /* if size is 0, caller used an invalid Huffman table entry */
    if (size == 0)
        ERREXIT(state->cinfo, JERR_HUFF_MISSING_CODE);

    put_buffer &= (((INT32) 1) << size) - 1; /* mask off any extra bits in code */
    put_bits += size; /* new number of bits in buffer */

    put_buffer <= 24 - put_bits; /* align incoming bits */
    put_buffer |= state->cur.put_buffer; /* and merge with old buffer contents */
}

```

```

while (put_bits >= 8) {
    int c = (int) ((put_buffer >> 16) & 0xFF);

    emit_byte(state, c, return FALSE);
    if (c == 0xFF) { /* need to stuff a zero byte? */
        emit_byte(state, 0, return FALSE);
    }
    put_buffer <<= 8;
    put_bits -= 8;
}

state->cur.put_buffer = put_buffer; /* update state variables */
state->cur.put_bits = put_bits;

return TRUE;
}

```

```

LOCAL(boolean)
flush_bits (working_state * state)
{
    if (! emit_bits(state, 0x7F, 7)) /* fill any partial byte with ones */
        return FALSE;
    state->cur.put_buffer = 0; /* and reset bit-buffer to empty */
    state->cur.put_bits = 0;
    return TRUE;
}

```

/\* Encode a single block's worth of coefficients \*/

```

LOCAL(boolean)
encode_one_block (working_state * state, JCOEFPTR block, int last_dc_val,
                  c_derived_tbl *dctbl, c_derived_tbl *actbl)
{
    register int temp, temp2;
    register int nbits;
    register int k, r, i;

    /* Encode the DC coefficient difference per section F.1.2.1 */
    temp = temp2 = block[0] - last_dc_val;

    if (temp < 0) {
        temp = -temp; /* temp is abs value of input */
        /* For a negative input, want temp2 = bitwise complement of abs(input) */
        /* This code assumes we are on a two's complement machine */
        temp2--;
    }

    /* Find the number of bits needed for the magnitude of the coefficient */
    nbits = 0;
    while (temp) {
        nbits++;
        temp >>= 1;
    }
    /* Check for out-of-range coefficient values.
     * Since we're encoding a difference, the range limit is twice as much.
     */
    if (nbits > MAX_COEF_BITS+1)
        ERREXIT(state->cinfo, JERR_BAD_DCT_COEF);

    /* Emit the Huffman-coded symbol for the number of bits */
    if (! emit_bits(state, dctbl->ehufco[nbits], dctbl->ehufsi[nbits]))
        return FALSE;

    /* Emit that number of bits of the value, if positive, */
    /* or the complement of its magnitude, if negative. */
    if (nbits) /* emit_bits rejects calls with size 0 */
        if (! emit_bits(state, (unsigned int) temp2, nbits))
            return FALSE;

    /* Encode the AC coefficients per section F.1.2.2 */

    r = 0; /* r = run length of zeros */

    for (k = 1; k < DCTSIZE2; k++) {
        if ((temp = block[jpeg_natural_order[k]]) == 0) {
            r++;
        } else {

```

```

/* if run length > 15, must emit special run-length-16 codes (0xF0) */
while (r > 15) {
if (! emit_bits(state, actbl->ehufco[0xF0], actbl->ehufsi[0xF0]))
return FALSE;
r -= 16;
}

temp2 = temp;
if (temp < 0) {
temp = -temp; /* temp is abs value of input */
/* This code assumes we are on a two's complement machine */
temp2--;
}

/* Find the number of bits needed for the magnitude of the coefficient */
nbits = 1; /* there must be at least one 1 bit */
while ((temp >= 1))
nbits++;
/* Check for out-of-range coefficient values */
if (nbits > MAX_COEF_BITS)
ERREXIT(state->cinfo, JERR_BAD_DCT_COEF);

/* Emit Huffman symbol for run length / number of bits */
i = (r < 4) + nbits;
if (! emit_bits(state, actbl->ehufco[i], actbl->ehufsi[i]))
return FALSE;

/* Emit that number of bits of the value, if positive, */
/* or the complement of its magnitude, if negative. */
if (! emit_bits(state, (unsigned int) temp2, nbits))
return FALSE;

r = 0;
}

/* If the last coef(s) were zero, emit an end-of-block code */
if (r > 0)
if (! emit_bits(state, actbl->ehufco[0], actbl->ehufsi[0]))
return FALSE;
return TRUE;
}

/* Emit a restart marker & resynchronize predictions.
*/
LOCAL(boolean)
emit_restart (working_state * state, int restart_num)
{
int ci;

if (! flush_bits(state))
return FALSE;

emit_byte(state, 0xFF, return FALSE);
emit_byte(state, JPEG_RST0 + restart_num, return FALSE);

/* Re-initialize DC predictions to 0 */
for (ci = 0; ci < state->cinfo->comps_in_scan; ci++)
state->cur.last_dc_val[ci] = 0;

/* The restart counter is not updated until we successfully write the MCU. */

return TRUE;
}

/*
* Encode and output one MCU's worth of Huffman-compressed coefficients.
*/

METHODDEF(boolean)
encode_mcu_huff (j_compress_ptr cinfo, JBLOCKROW *MCU_data)
{
huff_entropy_ptr entropy = (huff_entropy_ptr) cinfo->entropy;
working_state state;
int blkn, ci;

```

```

jpeg_component_info * compptr;

/* Load up working state */
state.next_output_byte = cinfo->dest->next_output_byte;
state.free_in_buffer = cinfo->dest->free_in_buffer;
ASSIGN_STATE(state.cur, entropy->saved);
state.cinfo = cinfo;

/* Emit restart marker if needed */
if (cinfo->restart_interval) {
    if (entropy->restarts_to_go == 0)
        if (! emit_restart(&state, entropy->next_restart_num))
            return FALSE;
}

/* Encode the MCU data blocks */
for (blkn = 0; blkcn < cinfo->blocks_in_MCU; blkcn++) {
    ci = cinfo->MCU_membership[blkcn];
    compptr = cinfo->cur_comp_info[ci];
    if (! encode_one_block(&state,
        MCU_data[blkcn][0], state.cur.last_dc_val[ci],
        entropy->dc_derived_tbls[compptr->dc_tbl_no],
        entropy->ac_derived_tbls[compptr->ac_tbl_no]))
        return FALSE;
    /* Update last_dc_val */
    state.cur.last_dc_val[ci] = MCU_data[blkcn][0][0];
}

/* Completed MCU, so update state */
cinfo->dest->next_output_byte = state.next_output_byte;
cinfo->dest->free_in_buffer = state.free_in_buffer;
ASSIGN_STATE(entropy->saved, state.cur);

/* Update restart-interval state too */
if (cinfo->restart_interval) {
    if (entropy->restarts_to_go == 0) {
        entropy->restarts_to_go = cinfo->restart_interval;
        entropy->next_restart_num++;
        entropy->next_restart_num &= 7;
    }
    entropy->restarts_to_go--;
}

return TRUE;
}

/* Finish up at the end of a Huffman-compressed scan.
*/
METHODDEF(void)
finish_pass_huff (j_compress_ptr cinfo)
{
    huff_entropy_ptr entropy = (huff_entropy_ptr) cinfo->entropy;
    working_state state;

    /* Load up working state ... flush_bits needs it */
    state.next_output_byte = cinfo->dest->next_output_byte;
    state.free_in_buffer = cinfo->dest->free_in_buffer;
    ASSIGN_STATE(state.cur, entropy->saved);
    state.cinfo = cinfo;

    /* Flush out the last data */
    if (! flush_bits(&state))
        ERREXIT(cinfo, JERR_CANT_SUSPEND);

    /* Update state */
    cinfo->dest->next_output_byte = state.next_output_byte;
    cinfo->dest->free_in_buffer = state.free_in_buffer;
    ASSIGN_STATE(entropy->saved, state.cur);
}

/*
 * Huffman coding optimization.
 *
 * We first scan the supplied data and count the number of uses of each symbol
 * that is to be Huffman-coded. (This process MUST agree with the code above.)
 * Then we build a Huffman coding tree for the observed counts.

```

```

* Symbols which are not needed at all for the particular image are not
* assigned any code, which saves space in the DHT marker as well as in
* the compressed data.
*/

```

```

#ifdef ENTROPY_OPT_SUPPORTED

```

```

/* Process a single block's worth of coefficients */

```

```

LOCAL(void)

```

```

htest_one_block (j_compress_ptr cinfo, JCOEFPTR block, int last_dc_val,
                 long dc_counts[], long ac_counts[])

```

```

{
    register int temp;
    register int nbits;
    register int k, r;

```

```

    /* Encode the DC coefficient difference per section F.1.2.1 */

```

```

    temp = block[0] - last_dc_val;
    if (temp < 0)
        temp = -temp;

```

```

    /* Find the number of bits needed for the magnitude of the coefficient */
    nbits = 0;
    while (temp) {
        nbits++;
        temp >>= 1;
    }

```

```

    /* Check for out-of-range coefficient values.

```

```

    * Since we're encoding a difference, the range limit is twice as much.
    */

```

```

    if (nbits > MAX_COEF_BITS+1)
        ERREXIT(cinfo, JERR_BAD_DCT_COEF);

```

```

    /* Count the Huffman symbol for the number of bits */
    dc_counts[nbits]++;

```

```

    /* Encode the AC coefficients per section F.1.2.2 */

```

```

    r = 0;          /* r = run length of zeros */

```

```

    for (k = 1; k < DCTSIZE2; k++) {
        if ((temp = block[jpeg_natural_order[k]]) == 0) {
            r++;
        } else {
            /* if run length > 15, must emit special run-length-16 codes (0xF0) */
            while (r > 15) {
                ac_counts[0xF0]++;
                r -= 16;
            }

```

```

            /* Find the number of bits needed for the magnitude of the coefficient */
            if (temp < 0)
                temp = -temp;

```

```

            /* Find the number of bits needed for the magnitude of the coefficient */
            nbits = 1;          /* there must be at least one 1 bit */
            while ((temp >>= 1))
                nbits++;

```

```

            /* Check for out-of-range coefficient values */
            if (nbits > MAX_COEF_BITS)
                ERREXIT(cinfo, JERR_BAD_DCT_COEF);

```

```

            /* Count Huffman symbol for run length / number of bits */
            ac_counts[(r << 4) + nbits]++;

```

```

            r = 0;
        }
    }

```

```

    /* If the last coef(s) were zero, emit an end-of-block code */

```

```

    if (r > 0)
        ac_counts[0]++;
}

```

```

/*
* Trial-encode one MCU's worth of Huffman-compressed coefficients.

```

```

* No data is actually output, so no suspension return is possible.
*/

```

```

METHODDEF(boolean)
encode_mcu_gather (j_compress_ptr cinfo, JBLOCKROW *MCU_data)
{
    huff_entropy_ptr entropy = (huff_entropy_ptr) cinfo->entropy;
    int blkn, ci;
    jpeg_component_info * compptr;

    /* Take care of restart intervals if needed */
    if (cinfo->restart_interval) {
        if (entropy->restarts_to_go == 0) {
            /* Re-initialize DC predictions to 0 */
            for (ci = 0; ci < cinfo->comps_in_scan; ci++)
                entropy->saved_last_dc_val[ci] = 0;
            /* Update restart state */
            entropy->restarts_to_go = cinfo->restart_interval;
        }
        entropy->restarts_to_go--;
    }

    for (blkn = 0; blkn < cinfo->blocks_in_MCU; blkn++) {
        ci = cinfo->MCU_membership[blkn];
        compptr = cinfo->cur_comp_info[ci];
        htest_one_block(cinfo, MCU_data[blkn][0], entropy->saved_last_dc_val[ci],
            entropy->dc_count_ptrs[compptr->dc_tbl_no],
            entropy->ac_count_ptrs[compptr->ac_tbl_no]);
        entropy->saved_last_dc_val[ci] = MCU_data[blkn][0][0];
    }

    return TRUE;
}

```

```

/* Generate the best Huffman code table for the given counts, fill htbl.
* Note this is also used by jcpuff.c.
*/

```

```

The JPEG standard requires that no symbol be assigned a codeword of all
one bits (so that padding bits added at the end of a compressed segment
can't look like a valid code). Because of the canonical ordering of
codewords, this just means that there must be an unused slot in the
longest codeword length category. Section K.2 of the JPEG spec suggests
reserving such a slot by pretending that symbol 256 is a valid symbol
with count 1. In theory that's not optimal; giving it count zero but
including it in the symbol set anyway should give a better Huffman code.
But the theoretically better code actually seems to come out worse in
practice, because it produces more all-ones bytes (which incur stuffed
zero bytes in the final file). In any case the difference is tiny.

```

```

The JPEG standard requires Huffman codes to be no more than 16 bits long.
If some symbols have a very small but nonzero probability, the Huffman tree
must be adjusted to meet the code length restriction. We currently use
the adjustment method suggested in JPEG section K.2. This method is *not*
optimal; it may not choose the best possible limited-length code. But
typically only very-low-frequency symbols will be given less-than-optimal
lengths, so the code is almost optimal. Experimental comparisons against
an optimal limited-length-code algorithm indicate that the difference is
microscopic --- usually less than a hundredth of a percent of total size.
So the extra complexity of an optimal algorithm doesn't seem worthwhile.
*/

```

```

GLOBAL(void)
jpeg_gen_optimal_table (j_compress_ptr cinfo, JHUFF_TBL * htbl, long freq[])
{
#define MAX_CLEN 32 /* assumed maximum initial code length */
    UINT8 bits[MAX_CLEN+1]; /* bits[k] = # of symbols with code length k */
    int codesize[257]; /* codesize[k] = code length of symbol k */
    int others[257]; /* next symbol in current branch of tree */
    int c1, c2;
    int p, i, j;
    long v;

    /* This algorithm is explained in section K.2 of the JPEG standard */

    MEMZERO(bits, sizeof(bits));
    MEMZERO(codesize, sizeof(codesize));
    for (i = 0; i < 257; i++)
        others[i] = -1; /* init links to empty */
}

```



```

freq[256] = 1;          /* make sure 256 has a nonzero count */
/* Including the pseudo-symbol 256 in the Huffman procedure guarantees
 * that no real symbol is given code-value of all ones, because 256
 * will be placed last in the largest codeword category.
 */

/* Huffman's basic algorithm to assign optimal code lengths to symbols */

for (;;) {
    /* Find the smallest nonzero frequency, set c1 = its symbol */
    /* In case of ties, take the larger symbol number */
    c1 = -1;
    v = 1000000000L;
    for (i = 0; i <= 256; i++) {
        if (freq[i] && freq[i] <= v) {
            v = freq[i];
            c1 = i;
        }
    }

    /* Find the next smallest nonzero frequency, set c2 = its symbol */
    /* In case of ties, take the larger symbol number */
    c2 = -1;
    v = 1000000000L;
    for (i = 0; i <= 256; i++) {
        if (freq[i] && freq[i] <= v && i != c1) {
            v = freq[i];
            c2 = i;
        }
    }

    /* Done if we've merged everything into one frequency */
    if (c2 < 0)
        break;

    /* Else merge the two counts/trees */
    freq[c1] += freq[c2];
    freq[c2] = 0;

    /* Increment the codesize of everything in c1's tree branch */
    codesize[c1]++;
    while (others[c1] >= 0) {
        c1 = others[c1];
        codesize[c1]++;
    }

    others[c1] = c2;          /* chain c2 onto c1's tree branch */

    /* Increment the codesize of everything in c2's tree branch */
    codesize[c2]++;
    while (others[c2] >= 0) {
        c2 = others[c2];
        codesize[c2]++;
    }
}

/* Now count the number of symbols of each code length */
for (i = 0; i <= 256; i++) {
    if (codesize[i]) {
        /* The JPEG standard seems to think that this can't happen, */
        /* but I'm paranoid... */
        if (codesize[i] > MAX_CLEN)
            ERREXIT(cinfo, JERR_HUFF_CLEN_OVERFLOW);

        bits[codesize[i]]++;
    }
}

/* JPEG doesn't allow symbols with code lengths over 16 bits, so if the pure
 * Huffman procedure assigned any such lengths, we must adjust the coding.
 * Here is what the JPEG spec says about how this next bit works:
 * Since symbols are paired for the longest Huffman code, the symbols are
 * removed from this length category two at a time. The prefix for the pair
 * (which is one bit shorter) is allocated to one of the pair; then,
 * skipping the BITS entry for that prefix length, a code word from the next
 * shortest nonzero BITS entry is converted into a prefix for two code words
 * one bit longer.
 */

```

```

for (i = MAX_CLEN; i > 16; i--) {
    while (bits[i] > 0) {
        j = i - 2; /* find length of new prefix to be used */
        while (bits[j] == 0)
            j--;

        bits[i] -= 2; /* remove two symbols */
        bits[i-1]++; /* one goes in this length */
        bits[j+1] += 2; /* two new symbols in this length */
        bits[j]--; /* symbol of this length is now a prefix */
    }
}

/* Remove the count for the pseudo-symbol 256 from the largest codelength */
while (bits[i] == 0) /* find largest codelength still in use */
    i--;
bits[i]--;

/* Return final symbol counts (only for lengths 0..16) */
MEMCOPY(htbl->bits, bits, SIZEOF(htbl->bits));

/* Return a list of the symbols sorted by code length */
/* It's not real clear to me why we don't need to consider the codelength
 * changes made above, but the JPEG spec seems to think this works.
 */
p = 0;
for (i = 1; i <= MAX_CLEN; i++) {
    for (j = 0; j <= 255; j++) {
        if (codesize[j] == i) {
            htbl->huffval[p] = (UINT8) j;
            p++;
        }
    }
}

/* Set sent_table FALSE so updated table will be written to JPEG file. */
htbl->sent_table = FALSE;
}

/* Finish up a statistics-gathering pass and create the new Huffman tables.
 */
METHODDEF(void)
finish_pass_gather (j_compress_ptr cinfo)
{
    huff_entropy_ptr entropy = (huff_entropy_ptr) cinfo->entropy;
    int ci, dctbl, actbl;
    jpeg_component_info * compptr;
    JHUFF_TBL **htblptr;
    boolean did_dc[NUM_HUFF_TBLS];
    boolean did_ac[NUM_HUFF_TBLS];

    /* It's important not to apply jpeg_gen_optimal_table more than once
     * per table, because it clobbers the input frequency counts!
     */
    MEMZERO(did_dc, SIZEOF(did_dc));
    MEMZERO(did_ac, SIZEOF(did_ac));

    for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
        compptr = cinfo->cur_comp_info[ci];
        dctbl = compptr->dc_tbl_no;
        actbl = compptr->ac_tbl_no;
        if (! did_dc[dctbl]) {
            htblptr = & cinfo->dc_huff_tbl_ptrs[dctbl];
            if (*htblptr == NULL)
                *htblptr = jpeg_alloc_huff_table((j_common_ptr) cinfo);
            jpeg_gen_optimal_table(cinfo, *htblptr, entropy->dc_count_ptrs[dctbl]);
            did_dc[dctbl] = TRUE;
        }
        if (! did_ac[actbl]) {
            htblptr = & cinfo->ac_huff_tbl_ptrs[actbl];
            if (*htblptr == NULL)
                *htblptr = jpeg_alloc_huff_table((j_common_ptr) cinfo);
            jpeg_gen_optimal_table(cinfo, *htblptr, entropy->ac_count_ptrs[actbl]);
            did_ac[actbl] = TRUE;
        }
    }
}
}

```

```
#endif /* ENTROPY_OPT_SUPPORTED */
```

```
/*  
 * Module initialization routine for Huffman entropy encoding.  
 */
```

```
GLOBAL(void)
```

```
jinit_huff_encoder (j_compress_ptr cinfo)
```

```
{  
    huff_entropy_ptr entropy;  
    int i;  
  
    entropy = (huff_entropy_ptr)  
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,  
        sizeof(huff_entropy_encoder));  
    cinfo->entropy = (struct jpeg_entropy_encoder *) entropy;  
    entropy->pub.start_pass = start_pass_huff;  
  
    /* Mark tables unallocated */  
    for (i = 0; i < NUM_HUFF_TBLS; i++) {  
        entropy->dc_derived_tbls[i] = entropy->ac_derived_tbls[i] = NULL;  
#ifdef ENTROPY_OPT_SUPPORTED  
        entropy->dc_count_ptrs[i] = entropy->ac_count_ptrs[i] = NULL;  
#endif  
    }  
}
```

```

/*
 * jcinit.c
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains initialization logic for the JPEG compressor.
 * This routine is in charge of selecting the modules to be executed and
 * making an initialization call to each one.
 *
 * Logically, this code belongs in jcmaster.c. It's split out because
 * linking this routine implies linking the entire compression library.
 * For a transcoding-only application, we want to be able to use jcmaster.c
 * without linking in the whole library.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/*
 * Master selection of compression modules.
 * This is done once at the start of processing an image. We determine
 * which modules will be used and give them appropriate initialization calls.
 */

GLOBAL(void)
jinit_compress_master (j_compress_ptr cinfo)
{
  /* Initialize master control (includes parameter checking/processing) */
  jinit_c_master_control(cinfo, FALSE /* full compression */);

  /* Preprocessing */
  if (! cinfo->raw_data_in) {
    jinit_color_converter(cinfo);
    jinit_downsampler(cinfo);
    jinit_c_prep_controller(cinfo, FALSE /* never need full buffer here */);
  }

  /* Forward DCT */
  jinit_forward_dct(cinfo);

  /* Entropy encoding: either Huffman or arithmetic coding. */
  if (cinfo->arith_code) {
    ERREXIT(cinfo, JERR_ARITH_NOTIMPL);
  } else {
    if (cinfo->progressive_mode) {
#ifdef C_PROGRESSIVE_SUPPORTED
      jinit_phuff_encoder(cinfo);
    #else
      ERREXIT(cinfo, JERR_NOT_COMPILED);
    #endif
    } else
      jinit_huff_encoder(cinfo);
  }

  /* Need a full-image coefficient buffer in any multi-pass mode. */
  jinit_c_coef_controller(cinfo,
    (boolean) (cinfo->num_scans > 1 || cinfo->optimize_coding));
  jinit_c_main_controller(cinfo, FALSE /* never need full buffer here */);

  jinit_marker_writer(cinfo);

  /* We can now tell the memory manager to allocate virtual arrays. */
  (*cinfo->mem->realize_virt_arrays) ((j_common_ptr) cinfo);

  /* Write the datastream header (SOI) immediately.
   * Frame and scan headers are postponed till later.
   * This lets application insert special markers after the SOI.
   */
  (*cinfo->marker->write_file_header) (cinfo);
}

```

```

/*
 * jcmaintct.c
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains the main buffer controller for compression.
 * The main buffer lies between the pre-processor and the JPEG
 * compressor proper; it holds downsampled data in the JPEG colorspace.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/* Note: currently, there is no operating mode in which a full-image buffer
 * is needed at this step. If there were, that mode could not be used with
 * "raw data" input, since this module is bypassed in that case. However,
 * we've left the code here for possible use in special applications.
 */
#undef FULL_MAIN_BUFFER_SUPPORTED

/* Private buffer controller object */

typedef struct {
  struct jpeg_c_main_controller pub; /* public fields */

  JDIMENSION cur_imcu_row; /* number of current iMCU row */
  JDIMENSION rowgroup_ctr; /* counts row groups received in iMCU row */
  boolean suspended; /* remember if we suspended output */
  J_BUF_MODE pass_mode; /* current operating mode */

  /* If using just a strip buffer, this points to the entire set of buffers
   * (we allocate one for each component). In the full-image case, this
   * points to the currently accessible strips of the virtual arrays.
   */
  JSAMPARRAY buffer[MAX_COMPONENTS];

#ifdef FULL_MAIN_BUFFER_SUPPORTED
  /* If using full-image storage, this array holds pointers to virtual-array
   * control blocks for each component. Unused if not full-image storage.
   */
  jvirt_sarray_ptr whole_image[MAX_COMPONENTS];
#endif
} my_main_controller;

typedef my_main_controller * my_main_ptr;

/* Forward declarations */
METHODDEF(void) process_data_simple_main
  JPP((j_compress_ptr cinfo, JSAMPARRAY input_buf,
       JDIMENSION *in_row_ctr, JDIMENSION in_rows_avail));
#ifdef FULL_MAIN_BUFFER_SUPPORTED
METHODDEF(void) process_data_buffer_main
  JPP((j_compress_ptr cinfo, JSAMPARRAY input_buf,
       JDIMENSION *in_row_ctr, JDIMENSION in_rows_avail));
#endif

/*
 * Initialize for a processing pass.
 */

METHODDEF(void)
start_pass_main (j_compress_ptr cinfo, J_BUF_MODE pass_mode)
{
  my_main_ptr main = (my_main_ptr) cinfo->main;

  /* Do nothing in raw-data mode. */
  if (cinfo->raw_data_in)
    return;

  main->cur_imcu_row = 0; /* initialize counters */
  main->rowgroup_ctr = 0;
  main->suspended = FALSE;
  main->pass_mode = pass_mode; /* save mode for use by process_data */
}

```

```

switch (pass_mode) {
case JBUF_PASS_THRU:
#ifdef FULL_MAIN_BUFFER_SUPPORTED
    if (main->whole_image[0] != NULL)
        ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
#endif
    main->pub.process_data = process_data_simple_main;
    break;
#ifdef FULL_MAIN_BUFFER_SUPPORTED
case JBUF_SAVE_SOURCE:
case JBUF_CRANK_DEST:
case JBUF_SAVE_AND_PASS:
    if (main->whole_image[0] == NULL)
        ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
    main->pub.process_data = process_data_buffer_main;
    break;
#endif
default:
    ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
    break;
}
}

/*
 * Process some data.
 * This routine handles the simple pass-through mode,
 * where we have only a strip buffer.
 */

METHODDEF(void)
process_data_simple_main (j_compress_ptr cinfo,
                          JSAMPARRAY input_buf, JDIMENSION *in_row_ctr,
                          JDIMENSION in_rows_avail)
{
    my_main_ptr main = (my_main_ptr) cinfo->main;

    while (main->cur_iMCU_row < cinfo->total_iMCU_rows) {
        /* Read input data if we haven't filled the main buffer yet */
        if (main->rowgroup_ctr < DCTSIZE)
            (*cinfo->prep->pre_process_data) (cinfo,
                                              input_buf, in_row_ctr, in_rows_avail,
                                              main->buffer, &main->rowgroup_ctr,
                                              (JDIMENSION) DCTSIZE);

        /* If we don't have a full iMCU row buffered, return to application for
         * more data. Note that preprocessor will always pad to fill the iMCU row
         * at the bottom of the image.
         */
        if (main->rowgroup_ctr != DCTSIZE)
            return;

        /* Send the completed row to the compressor */
        if (! (*cinfo->coef->compress_data) (cinfo, main->buffer)) {
            /* If compressor did not consume the whole row, then we must need to
             * suspend processing and return to the application. In this situation
             * we pretend we didn't yet consume the last input row; otherwise, if
             * it happened to be the last row of the image, the application would
             * think we were done.
             */
            if (! main->suspended) {
                (*in_row_ctr)--;
                main->suspended = TRUE;
            }
            return;
        }
        /* We did finish the row. Undo our little suspension hack if a previous
         * call suspended; then mark the main buffer empty.
         */
        if (main->suspended) {
            (*in_row_ctr)++;
            main->suspended = FALSE;
        }
        main->rowgroup_ctr = 0;
        main->cur_iMCU_row++;
    }
}

```

```

#ifdef FULL_MAIN_BUFFER_SUPPORTED

/*
 * Process some data.
 * This routine handles all of the modes that use a full-size buffer.
 */

METHODDEF(void)
process_data_buffer_main (j_compress_ptr cinfo,
                          JSAMPARRAY input_buf, JDIMENSION *in_row_ctr,
                          JDIMENSION in_rows_avail)
{
    my_main_ptr main = (my_main_ptr) cinfo->main;
    int ci;
    jpeg_component_info *comp_ptr;
    boolean writing = (main->pass_mode != JBUF_CRANK_DEST);

    while (main->cur_iMCU_row < cinfo->total_iMCU_rows) {
        /* Realign the virtual buffers if at the start of an iMCU row. */
        if (main->rowgroup_ctr == 0) {
            for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
                 ci++, comp_ptr++) {
                main->buffer[ci] = (*cinfo->mem->access_virt_sarray)
                    ((j_common_ptr) cinfo, main->whole_image[ci],
                     main->cur_iMCU_row * (comp_ptr->v_samp_factor * DCTSIZE),
                     (JDIMENSION) (comp_ptr->v_samp_factor * DCTSIZE), writing);
            }
            /* In a read pass, pretend we just read some source data. */
            if (! writing) {
                *in_row_ctr += cinfo->max_v_samp_factor * DCTSIZE;
                main->rowgroup_ctr = DCTSIZE;
            }
        }

        /* If a write pass, read input data until the current iMCU row is full. */
        /* Note: preprocessor will pad if necessary to fill the last iMCU row. */
        if (writing) {
            (*cinfo->prep->pre_process_data) (cinfo,
                                              input_buf, in_row_ctr, in_rows_avail,
                                              main->buffer, &main->rowgroup_ctr,
                                              (JDIMENSION) DCTSIZE);
            /* Return to application if we need more data to fill the iMCU row. */
            if (main->rowgroup_ctr < DCTSIZE)
                return;
        }

        /* Emit data, unless this is a sink-only pass. */
        if (main->pass_mode != JBUF_SAVE_SOURCE) {
            if (! (*cinfo->coef->compress_data) (cinfo, main->buffer)) {
                /* If compressor did not consume the whole row, then we must need to
                 * suspend processing and return to the application. In this situation
                 * we pretend we didn't yet consume the last input row; otherwise, if
                 * it happened to be the last row of the image, the application would
                 * think we were done.
                 */
                if (! main->suspended) {
                    (*in_row_ctr)--;
                    main->suspended = TRUE;
                }
                return;
            }
            /* We did finish the row. Undo our little suspension hack if a previous
             * call suspended; then mark the main buffer empty.
             */
            if (main->suspended) {
                (*in_row_ctr)++;
                main->suspended = FALSE;
            }
        }

        /* If get here, we are done with this iMCU row. Mark buffer empty. */
        main->rowgroup_ctr = 0;
        main->cur_iMCU_row++;
    }
}

#endif /* FULL_MAIN_BUFFER_SUPPORTED */

/*

```

```

* Initialize main buffer controller.
*/

GLOBAL(void)
jinit_c_main_controller (j_compress_ptr cinfo, boolean need_full_buffer)
{
    my_main_ptr main;
    int ci;
    jpeg_component_info *comp_ptr;

    main = (my_main_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                                   sizeof(my_main_controller));
    cinfo->main = (struct jpeg_c_main_controller *) main;
    main->pub.start_pass = start_pass_main;

    /* We don't need to create a buffer in raw-data mode. */
    if (cinfo->raw_data_in)
        return;

    /* Create the buffer.  It holds downsampled data, so each component
     * may be of a different size.
     */
    if (need_full_buffer) {
#ifdef FULL_MAIN_BUFFER_SUPPORTED
        /* Allocate a full-image virtual array for each component */
        /* Note we pad the bottom to a multiple of the iMCU height */
        for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
             ci++, comp_ptr++) {
            main->whole_image[ci] = (*cinfo->mem->request_virt_sarray)
                ((j_common_ptr) cinfo, JPOOL_IMAGE, FALSE,
                 comp_ptr->width_in_blocks * DCTSIZE,
                 (JDIMENSION) jround_up((long) comp_ptr->height_in_blocks,
                                         (long) comp_ptr->v_samp_factor) * DCTSIZE,
                 (JDIMENSION) (comp_ptr->v_samp_factor * DCTSIZE));
        }
#else
        ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
#endif
    } else {
#ifdef FULL_MAIN_BUFFER_SUPPORTED
        main->whole_image[0] = NULL; /* flag for no virtual arrays */
#endif
        /* Allocate a strip buffer for each component */
        for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
             ci++, comp_ptr++) {
            main->buffer[ci] = (*cinfo->mem->alloc_sarray)
                ((j_common_ptr) cinfo, JPOOL_IMAGE,
                 comp_ptr->width_in_blocks * DCTSIZE,
                 (JDIMENSION) (comp_ptr->v_samp_factor * DCTSIZE));
        }
    }
}

```



```

/*
 * jcmarker.c
 *
 * Copyright (C) 1991-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains routines to write JPEG datastream markers.
 */

```

```

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

```

```

typedef enum {                /* JPEG marker codes */
  M_SOF0  = 0xc0,
  M_SOF1  = 0xc1,
  M_SOF2  = 0xc2,
  M_SOF3  = 0xc3,

  M_SOF5  = 0xc5,
  M_SOF6  = 0xc6,
  M_SOF7  = 0xc7,

  M_JPG   = 0xc8,
  M_SOF9  = 0xc9,
  M_SOF10 = 0xca,
  M_SOF11 = 0xcb,

  M_SOF13 = 0xcd,
  M_SOF14 = 0xce,
  M_SOF15 = 0xcf,

  M_DHT   = 0xcc,

  M_DAC   = 0xcc,

  M_RST0  = 0xd0,
  M_RST1  = 0xd1,
  M_RST2  = 0xd2,
  M_RST3  = 0xd3,
  M_RST4  = 0xd4,
  M_RST5  = 0xd5,
  M_RST6  = 0xd6,
  M_RST7  = 0xd7,

  M_SOI   = 0xd8,
  M_EOI   = 0xd9,
  M_SOS   = 0xda,
  M_DQT   = 0xdb,
  M_DNL   = 0xdc,
  M_DRI   = 0xdd,
  M_DHP   = 0xde,
  M_EXP   = 0xdf,

  M_APP0  = 0xe0,
  M_APP1  = 0xe1,
  M_APP2  = 0xe2,
  M_APP3  = 0xe3,
  M_APP4  = 0xe4,
  M_APP5  = 0xe5,
  M_APP6  = 0xe6,
  M_APP7  = 0xe7,
  M_APP8  = 0xe8,
  M_APP9  = 0xe9,
  M_APP10 = 0xea,
  M_APP11 = 0xeb,
  M_APP12 = 0xec,
  M_APP13 = 0xed,
  M_APP14 = 0xee,
  M_APP15 = 0xef,

  M_JPG0  = 0xf0,
  M_JPG13 = 0xfd,
  M_COM   = 0xfe,

  M_TEM   = 0x01,

  M_ERROR = 0x100

```

```

} JPEG_MARKER;

/* Private state */

typedef struct {
    struct jpeg_marker_writer pub; /* public fields */

    unsigned int last_restart_interval; /* last DRI value emitted; 0 after SOI */
} my_marker_writer;

typedef my_marker_writer * my_marker_ptr;

/*
 * Basic output routines.
 *
 * Note that we do not support suspension while writing a marker.
 * Therefore, an application using suspension must ensure that there is
 * enough buffer space for the initial markers (typ. 600-700 bytes) before
 * calling jpeg_start_compress, and enough space to write the trailing EOI
 * (a few bytes) before calling jpeg_finish_compress. Multipass compression
 * modes are not supported at all with suspension, so those two are the only
 * points where markers will be written.
 */

LOCAL(void)
emit_byte (j_compress_ptr cinfo, int val)
/* Emit a byte */
{
    struct jpeg_destination_mgr * dest = cinfo->dest;

    (dest->next_output_byte)++ = (JOCTET) val;
    if (--dest->free_in_buffer == 0) {
        if (! (*dest->empty_output_buffer) (cinfo))
            ERREXIT(cinfo, JERR_CANT_SUSPEND);
    }
}

LOCAL(void)
emit_marker (j_compress_ptr cinfo, JPEG_MARKER mark)
/* Emit a marker code */
{
    emit_byte(cinfo, 0xFF);
    emit_byte(cinfo, (int) mark);
}

LOCAL(void)
emit_2bytes (j_compress_ptr cinfo, int value)
/* Emit a 2-byte integer; these are always MSB first in JPEG files */
{
    emit_byte(cinfo, (value >> 8) & 0xFF);
    emit_byte(cinfo, value & 0xFF);
}

/*
 * Routines to write specific marker types.
 */

LOCAL(int)
emit_dqt (j_compress_ptr cinfo, int index)
/* Emit a DQT marker */
/* Returns the precision used (0 = 8bits, 1 = 16bits) for baseline checking */
{
    JQUANT_TBL * qtbl = cinfo->quant_tbl_ptrs[index];
    int prec;
    int i;

    if (qtbl == NULL)
        ERREXIT1(cinfo, JERR_NO_QUANT_TABLE, index);

    prec = 0;
    for (i = 0; i < DCTSIZE2; i++) {
        if (qtbl->quantval[i] > 255)
            prec = 1;
    }
}

```

```

if (! qtbl->sent_table) {
    emit_marker(cinfo, M_DQT);

    emit_2bytes(cinfo, prec ? DCTSIZE2*2 + 1 + 2 : DCTSIZE2 + 1 + 2);

    emit_byte(cinfo, index + (prec<<4));

    for (i = 0; i < DCTSIZE2; i++) {
        /* The table entries must be emitted in zigzag order. */
        unsigned int qval = qtbl->quantval[jpeg_natural_order[i]];
        if (prec)
            emit_byte(cinfo, (int) (qval >> 8));
        emit_byte(cinfo, (int) (qval & 0xFF));
    }

    qtbl->sent_table = TRUE;
}

return prec;
}

```

```

LOCAL(void)
emit_dht (j_compress_ptr cinfo, int index, boolean is_ac)
/* Emit a DHT marker */
{
    JHUFF_TBL * htbl;
    int length, i;

    if (is_ac) {
        htbl = cinfo->ac_huff_tbl_ptrs[index];
        index += 0x10; /* output index has AC bit set */
    } else {
        htbl = cinfo->dc_huff_tbl_ptrs[index];
    }

    if (htbl == NULL)
        ERREXIT1(cinfo, JERR_NO_HUFF_TABLE, index);

    if (! htbl->sent_table) {
        emit_marker(cinfo, M_DHT);

        length = 0;
        for (i = 1; i <= 16; i++)
            length += htbl->bits[i];

        emit_2bytes(cinfo, length + 2 + 1 + 16);
        emit_byte(cinfo, index);

        for (i = 1; i <= 16; i++)
            emit_byte(cinfo, htbl->bits[i]);

        for (i = 0; i < length; i++)
            emit_byte(cinfo, htbl->huffval[i]);

        htbl->sent_table = TRUE;
    }
}

```

```

LOCAL(void)
emit_dac (j_compress_ptr cinfo)
/* Emit a DAC marker */
/* Since the useful info is so small, we want to emit all the tables in */
/* one DAC marker. Therefore this routine does its own scan of the table. */
{
#ifdef C_ARITH_CODING_SUPPORTED
    char dc_in_use[NUM_ARITH_TBLS];
    char ac_in_use[NUM_ARITH_TBLS];
    int length, i;
    jpeg_component_info *comp_ptr;

    for (i = 0; i < NUM_ARITH_TBLS; i++)
        dc_in_use[i] = ac_in_use[i] = 0;

    for (i = 0; i < cinfo->comps_in_scan; i++) {
        comp_ptr = cinfo->cur_comp_info[i];
        dc_in_use[comp_ptr->dc_tbl_no] = 1;
        ac_in_use[comp_ptr->ac_tbl_no] = 1;
    }
}

```

```

length = 0;
for (i = 0; i < NUM_ARITH_TBLS; i++)
    length += dc_in_use[i] + ac_in_use[i];

emit_marker(cinfo, M_DAC);

emit_2bytes(cinfo, length*2 + 2);

for (i = 0; i < NUM_ARITH_TBLS; i++) {
    if (dc_in_use[i]) {
        emit_byte(cinfo, i);
        emit_byte(cinfo, cinfo->arith_dc_L[i] + (cinfo->arith_dc_U[i]<<4));
    }
    if (ac_in_use[i]) {
        emit_byte(cinfo, i + 0x10);
        emit_byte(cinfo, cinfo->arith_ac_K[i]);
    }
}
#endif /* C_ARITH_CODING_SUPPORTED */
}

LOCAL(void)
emit_dri (j_compress_ptr cinfo)
/* Emit a DRI marker */
{
    emit_marker(cinfo, M_DRI);

    emit_2bytes(cinfo, 4);    /* fixed length */

    emit_2bytes(cinfo, (int) cinfo->restart_interval);
}

LOCAL(void)
emit_sof (j_compress_ptr cinfo, JPEG_MARKER code)
/* Emit a SOF marker */
{
    int ci;
    jpeg_component_info *comp_ptr;

    emit_marker(cinfo, code);

    emit_2bytes(cinfo, 3 * cinfo->num_components + 2 + 5 + 1); /* length */

    /* Make sure image isn't bigger than SOF field can handle */
    if ((long) cinfo->image_height > 65535L ||
        (long) cinfo->image_width > 65535L)
        ERREXIT1(cinfo, JERR_IMAGE_TOO_BIG, (unsigned int) 65535);

    emit_byte(cinfo, cinfo->data_precision);
    emit_2bytes(cinfo, (int) cinfo->image_height);
    emit_2bytes(cinfo, (int) cinfo->image_width);

    emit_byte(cinfo, cinfo->num_components);

    for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
         ci++, comp_ptr++) {
        emit_byte(cinfo, comp_ptr->component_id);
        emit_byte(cinfo, (comp_ptr->h_samp_factor << 4) + comp_ptr->v_samp_factor);
        emit_byte(cinfo, comp_ptr->quant_tbl_no);
    }
}

LOCAL(void)
emit_sos (j_compress_ptr cinfo)
/* Emit a SOS marker */
{
    int i, td, ta;
    jpeg_component_info *comp_ptr;

    emit_marker(cinfo, M_SOS);

    emit_2bytes(cinfo, 2 * cinfo->comps_in_scan + 2 + 1 + 3); /* length */

    emit_byte(cinfo, cinfo->comps_in_scan);

    for (i = 0; i < cinfo->comps_in_scan; i++) {

```

```

    compptr = cinfo->cur_comp_info[i];
    emit_byte(cinfo, compptr->component_id);
    td = compptr->dc_tbl_no;
    ta = compptr->ac_tbl_no;
    if (cinfo->progressive_mode) {
        /* Progressive mode: only DC or only AC tables are used in one scan;
         * furthermore, Huffman coding of DC refinement uses no table at all.
         * We emit 0 for unused field(s); this is recommended by the P&M text
         * but does not seem to be specified in the standard.
         */
        if (cinfo->Ss == 0) {
            ta = 0;          /* DC scan */
        }
        if (cinfo->Ah != 0 && !cinfo->arith_code)
            td = 0;          /* no DC table either */
        else {
            td = 0;          /* AC scan */
        }
    }
    emit_byte(cinfo, (td << 4) + ta);
}

emit_byte(cinfo, cinfo->Ss);
emit_byte(cinfo, cinfo->Se);
emit_byte(cinfo, (cinfo->Ah << 4) + cinfo->Al);
}

```

```

LOCAL(void)
emit_jfif_app0 (j_compress_ptr cinfo)
/* Emit a JFIF-compliant APP0 marker */
{
    /*
     * Length of APP0 block (2 bytes)
     * Block ID (4 bytes - ASCII "JFIF")
     * Zero byte (1 byte to terminate the ID string)
     * Version Major, Minor (2 bytes - major first)
     * Units (1 byte - 0x00 = none, 0x01 = inch, 0x02 = cm)
     * Xdpi (2 bytes - dots per unit horizontal)
     * Ydpi (2 bytes - dots per unit vertical)
     * Thumbnail X size (1 byte)
     * Thumbnail Y size (1 byte)
     */
    emit_marker(cinfo, M_APP0);
    emit_2bytes(cinfo, 2 + 4 + 1 + 2 + 1 + 2 + 2 + 1 + 1); /* length */
    emit_byte(cinfo, 0x4A); /* Identifier: ASCII "JFIF" */
    emit_byte(cinfo, 0x46);
    emit_byte(cinfo, 0x49);
    emit_byte(cinfo, 0x46);
    emit_byte(cinfo, 0);
    emit_byte(cinfo, cinfo->JFIF_major_version); /* Version fields */
    emit_byte(cinfo, cinfo->JFIF_minor_version);
    emit_byte(cinfo, cinfo->density_unit); /* Pixel size information */
    emit_2bytes(cinfo, (int) cinfo->X_density);
    emit_2bytes(cinfo, (int) cinfo->Y_density);
    emit_byte(cinfo, 0); /* No thumbnail image */
    emit_byte(cinfo, 0);
}

```

```

LOCAL(void)
emit_adobe_app14 (j_compress_ptr cinfo)
/* Emit an Adobe APP14 marker */
{
    /*
     * Length of APP14 block (2 bytes)
     * Block ID (5 bytes - ASCII "Adobe")
     * Version Number (2 bytes - currently 100)
     * Flags0 (2 bytes - currently 0)
     * Flags1 (2 bytes - currently 0)
     * Color transform (1 byte)
     *
     * Although Adobe TN 5116 mentions Version = 101, all the Adobe files
     * now in circulation seem to use Version = 100, so that's what we write.
     *
     * We write the color transform byte as 1 if the JPEG color space is
     * YCbCr, 2 if it's YCCK, 0 otherwise. Adobe's definition has to do with
     * whether the encoder performed a transformation, which is pretty useless.
     */
}

```

```

*/

emit_marker(cinfo, M_APP14);

emit_2bytes(cinfo, 2 + 5 + 2 + 2 + 2 + 1); /* length */

emit_byte(cinfo, 0x41); /* Identifier: ASCII "Adobe" */
emit_byte(cinfo, 0x64);
emit_byte(cinfo, 0x6F);
emit_byte(cinfo, 0x62);
emit_byte(cinfo, 0x65);
emit_2bytes(cinfo, 100); /* Version */
emit_2bytes(cinfo, 0); /* Flags0 */
emit_2bytes(cinfo, 0); /* Flags1 */
switch (cinfo->jpeg_color_space) {
case JCS_YCbCr:
    emit_byte(cinfo, 1); /* Color transform = 1 */
    break;
case JCS_YCCK:
    emit_byte(cinfo, 2); /* Color transform = 2 */
    break;
default:
    emit_byte(cinfo, 0); /* Color transform = 0 */
    break;
}
}

/*
 * These routines allow writing an arbitrary marker with parameters.
 * The only intended use is to emit COM or APPn markers after calling
 * write_file_header and before calling write_frame_header.
 * Other uses are not guaranteed to produce desirable results.
 * Counting the parameter bytes properly is the caller's responsibility.
 */

METHODDEF(void)
write_marker_header (j_compress_ptr cinfo, int marker, unsigned int datalen)
/* Emit an arbitrary marker header */
{
    if (datalen > (unsigned int) 65533) /* safety check */
        ERREXIT(cinfo, JERR_BAD_LENGTH);

    emit_marker(cinfo, (JPEG_MARKER) marker);

    emit_2bytes(cinfo, (int) (datalen + 2)); /* total length */
}

METHODDEF(void)
write_marker_byte (j_compress_ptr cinfo, int val)
/* Emit one byte of marker parameters following write_marker_header */
{
    emit_byte(cinfo, val);
}

/*
 * Write datastream header.
 * This consists of an SOI and optional APPn markers.
 * We recommend use of the JFIF marker, but not the Adobe marker,
 * when using YCbCr or grayscale data. The JFIF marker should NOT
 * be used for any other JPEG colorspace. The Adobe marker is helpful
 * to distinguish RGB, CMYK, and YCCK colorspaces.
 * Note that an application can write additional header markers after
 * jpeg_start_compress returns.
 */

METHODDEF(void)
write_file_header (j_compress_ptr cinfo)
{
    my_marker_ptr marker = (my_marker_ptr) cinfo->marker;

    emit_marker(cinfo, M_SOI); /* first the SOI */

    /* SOI is defined to reset restart interval to 0 */
    marker->last_restart_interval = 0;

    if (cinfo->write_JFIF_header) /* next an optional JFIF APP0 */
        emit_jfif_app0(cinfo);
    if (cinfo->write_Adobe_marker) /* next an optional Adobe APP14 */

```

```

    emit_adobe_app14(cinfo);
}

/*
 * Write frame header.
 * This consists of DQT and SOFn markers.
 * Note that we do not emit the SOF until we have emitted the DQT(s).
 * This avoids compatibility problems with incorrect implementations that
 * try to error-check the quant table numbers as soon as they see the SOF.
 */

METHODDEF(void)
write_frame_header (j_compress_ptr cinfo)
{
    int ci, prec;
    boolean is_baseline;
    jpeg_component_info *comp_ptr;

    /* Emit DQT for each quantization table.
     * Note that emit_dqt() suppresses any duplicate tables.
     */
    prec = 0;
    for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
         ci++, comp_ptr++) {
        prec += emit_dqt(cinfo, comp_ptr->quant_tbl_no);
    }
    /* now prec is nonzero iff there are any 16-bit quant tables. */

    /* Check for a non-baseline specification.
     * Note we assume that Huffman table numbers won't be changed later.
     */
    if (cinfo->arith_code || cinfo->progressive_mode ||
        cinfo->data_precision != 8) {
        is_baseline = FALSE;
    } else {
        is_baseline = TRUE;
        for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
             ci++, comp_ptr++) {
            if (comp_ptr->dc_tbl_no > 1 || comp_ptr->ac_tbl_no > 1)
                is_baseline = FALSE;
        }
        if (prec && is_baseline) {
            is_baseline = FALSE;
            /* If it's baseline except for quantizer size, warn the user */
            TRACEMS(cinfo, 0, JTRC_16BIT_TABLES);
        }
    }

    /* Emit the proper SOF marker */
    if (cinfo->arith_code) {
        emit_sof(cinfo, M_SOF9); /* SOF code for arithmetic coding */
    } else {
        if (cinfo->progressive_mode)
            emit_sof(cinfo, M_SOF2); /* SOF code for progressive Huffman */
        else if (is_baseline)
            emit_sof(cinfo, M_SOF0); /* SOF code for baseline implementation */
        else
            emit_sof(cinfo, M_SOF1); /* SOF code for non-baseline Huffman file */
    }
}

/*
 * Write scan header.
 * This consists of DHT or DAC markers, optional DRI, and SOS.
 * Compressed data will be written following the SOS.
 */

METHODDEF(void)
write_scan_header (j_compress_ptr cinfo)
{
    my_marker_ptr marker = (my_marker_ptr) cinfo->marker;
    int i;
    jpeg_component_info *comp_ptr;

    if (cinfo->arith_code) {
        /* Emit arith conditioning info. We may have some duplication
         * if the file has multiple scans, but it's so small it's hardly
         * worth worrying about.

```

```

    */
    emit_dac(cinfo);
} else {
    /* Emit Huffman tables.
     * Note that emit_dht() suppresses any duplicate tables.
     */
    for (i = 0; i < cinfo->comps_in_scan; i++) {
        compptr = cinfo->cur_comp_info[i];
        if (cinfo->progressive_mode) {
            /* Progressive mode: only DC or only AC tables are used in one scan */
            if (cinfo->Ss == 0) {
                if (cinfo->Ah == 0) /* DC needs no table for refinement scan */
                    emit_dht(cinfo, compptr->dc_tbl_no, FALSE);
            } else {
                emit_dht(cinfo, compptr->ac_tbl_no, TRUE);
            }
        } else {
            /* Sequential mode: need both DC and AC tables */
            emit_dht(cinfo, compptr->dc_tbl_no, FALSE);
            emit_dht(cinfo, compptr->ac_tbl_no, TRUE);
        }
    }
}

/* Emit DRI if required --- note that DRI value could change for each scan.
 * We avoid wasting space with unnecessary DRIs, however.
 */
if (cinfo->restart_interval != marker->last_restart_interval) {
    emit_dri(cinfo);
    marker->last_restart_interval = cinfo->restart_interval;
}

emit_sos(cinfo);
}

/*
 * Write datastream trailer.
 */
METHODDEF(void)
write_file_trailer (j_compress_ptr cinfo)
{
    emit_marker(cinfo, M_EOI);
}

/*
 * Write an abbreviated table-specification datastream.
 * This consists of SOI, DQT and DHT tables, and EOI.
 * Any table that is defined and not marked sent_table = TRUE will be
 * emitted. Note that all tables will be marked sent_table = TRUE at exit.
 */
METHODDEF(void)
write_tables_only (j_compress_ptr cinfo)
{
    int i;

    emit_marker(cinfo, M_SOI);

    for (i = 0; i < NUM_QUANT_TBLS; i++) {
        if (cinfo->quant_tbl_ptrs[i] != NULL)
            (void) emit_dqt(cinfo, i);
    }

    if (! cinfo->arith_code) {
        for (i = 0; i < NUM_HUFF_TBLS; i++) {
            if (cinfo->dc_huff_tbl_ptrs[i] != NULL)
                emit_dht(cinfo, i, FALSE);
            if (cinfo->ac_huff_tbl_ptrs[i] != NULL)
                emit_dht(cinfo, i, TRUE);
        }
    }

    emit_marker(cinfo, M_EOI);
}

/*

```





```

/*
 * jcmaster.c
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains master control logic for the JPEG compressor.
 * These routines are concerned with parameter validation, initial setup,
 * and inter-pass control (determining the number of passes and the work
 * to be done in each pass).
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/* Private state */

typedef enum {
    main_pass,          /* input data, also do first output step */
    huff_opt_pass,      /* Huffman code optimization pass */
    output_pass         /* data output pass */
} c_pass_type;

typedef struct {
    struct jpeg_comp_master pub; /* public fields */

    c_pass_type pass_type;      /* the type of the current pass */

    int pass_number;           /* # of passes completed */
    int total_passes;         /* total # of passes needed */

    int scan_number;          /* current index in scan_info[] */
} my_comp_master;

typedef my_comp_master * my_master_ptr;

/*
 * Support routines that do various essential calculations.
 */

LOCAL(void)
initial_setup (j_compress_ptr cinfo)
/* Do computations that are needed before master selection phase */
{
    int ci;
    jpeg_component_info *comp_ptr;
    long samplesperrow;
    JDIMENSION jd_samplesperrow;

    /* Sanity check on image dimensions */
    if (cinfo->image_height <= 0 || cinfo->image_width <= 0
        || cinfo->num_components <= 0 || cinfo->input_components <= 0)
        ERREXIT(cinfo, JERR_EMPTY_IMAGE);

    /* Make sure image isn't bigger than I can handle */
    if ((long) cinfo->image_height > (long) JPEG_MAX_DIMENSION ||
        (long) cinfo->image_width > (long) JPEG_MAX_DIMENSION)
        ERREXIT1(cinfo, JERR_IMAGE_TOO_BIG, (unsigned int) JPEG_MAX_DIMENSION);

    /* Width of an input scanline must be representable as JDIMENSION. */
    samplesperrow = (long) cinfo->image_width * (long) cinfo->input_components;
    jd_samplesperrow = (JDIMENSION) samplesperrow;
    if ((long) jd_samplesperrow != samplesperrow)
        ERREXIT(cinfo, JERR_WIDTH_OVERFLOW);

    /* For now, precision must match compiled-in value... */
    if (cinfo->data_precision != BITS_IN_JSAMPLE)
        ERREXIT1(cinfo, JERR_BAD_PRECISION, cinfo->data_precision);

    /* Check that number of components won't exceed internal array sizes */
    if (cinfo->num_components > MAX_COMPONENTS)
        ERREXIT2(cinfo, JERR_COMPONENT_COUNT, cinfo->num_components,
            MAX_COMPONENTS);

    /* Compute maximum sampling factors; check factor validity */
    cinfo->max_h_samp_factor = 1;

```

```

cinfo->max_v_samp_factor = 1;
for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
    ci++, compptr++) {
    if (compptr->h_samp_factor<=0 || compptr->h_samp_factor>MAX_SAMP_FACTOR ||
        compptr->v_samp_factor<=0 || compptr->v_samp_factor>MAX_SAMP_FACTOR)
        ERREXIT(cinfo, JERR_BAD_SAMPLING);
    cinfo->max_h_samp_factor = MAX(cinfo->max_h_samp_factor,
        compptr->h_samp_factor);
    cinfo->max_v_samp_factor = MAX(cinfo->max_v_samp_factor,
        compptr->v_samp_factor);
}

/* Compute dimensions of components */
for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
    ci++, compptr++) {
    /* Fill in the correct component_index value; don't rely on application */
    compptr->component_index = ci;
    /* For compression, we never do DCT scaling. */
    compptr->DCT_scaled_size = DCTSIZE;
    /* Size in DCT blocks */
    compptr->width_in_blocks = (JDIMENSION)
        jdiv_round_up((long) cinfo->image_width * (long) compptr->h_samp_factor,
            (long) (cinfo->max_h_samp_factor * DCTSIZE));
    compptr->height_in_blocks = (JDIMENSION)
        jdiv_round_up((long) cinfo->image_height * (long) compptr->v_samp_factor,
            (long) (cinfo->max_v_samp_factor * DCTSIZE));
    /* Size in samples */
    compptr->downsampled_width = (JDIMENSION)
        jdiv_round_up((long) cinfo->image_width * (long) compptr->h_samp_factor,
            (long) cinfo->max_h_samp_factor);
    compptr->downsampled_height = (JDIMENSION)
        jdiv_round_up((long) cinfo->image_height * (long) compptr->v_samp_factor,
            (long) cinfo->max_v_samp_factor);
    /* Mark component needed (this flag isn't actually used for compression) */
    compptr->component_needed = TRUE;

    /* Compute number of fully interleaved MCU rows (number of times that
     * main controller will call coefficient controller).
     */
    cinfo->total_iMCU_rows = (JDIMENSION)
        jdiv_round_up((long) cinfo->image_height,
            (long) (cinfo->max_v_samp_factor*DCTSIZE));
}

#ifdef C_MULTISCAN_FILES_SUPPORTED
LOCAL(void)
validate_script (j_compress_ptr cinfo)
/* Verify that the scan script in cinfo->scan_info[] is valid; also
 * determine whether it uses progressive JPEG, and set cinfo->progressive_mode.
 */
{
    const jpeg_scan_info * scanptr;
    int scanno, ncomps, ci, coefi, thisi;
    int Ss, Se, Ah, Al;
    boolean component_sent[MAX_COMPONENTS];
#ifdef C_PROGRESSIVE_SUPPORTED
    int * last_bitpos_ptr;
    int last_bitpos[MAX_COMPONENTS][DCTSIZE2];
    /* -1 until that coefficient has been seen; then last Al for it */
#endif

    if (cinfo->num_scans <= 0)
        ERREXIT1(cinfo, JERR_BAD_SCAN_SCRIPT, 0);

    /* For sequential JPEG, all scans must have Ss=0, Se=DCTSIZE2-1;
     * for progressive JPEG, no scan can have this.
     */
    scanptr = cinfo->scan_info;
    if (scanptr->Ss != 0 || scanptr->Se != DCTSIZE2-1) {
#ifdef C_PROGRESSIVE_SUPPORTED
        cinfo->progressive_mode = TRUE;
        last_bitpos_ptr = &last_bitpos[0][0];
        for (ci = 0; ci < cinfo->num_components; ci++)
            for (coefi = 0; coefi < DCTSIZE2; coefi++)
                *last_bitpos_ptr++ = -1;
#else
        ERREXIT(cinfo, JERR_NOT_COMPILED);
#endif
    }
}

```

```

#endif
} else {
    cinfo->progressive_mode = FALSE;
    for (ci = 0; ci < cinfo->num_components; ci++)
        component_sent[ci] = FALSE;
}

for (scanno = 1; scanno <= cinfo->num_scans; scanptr++, scanno++) {
    /* Validate component indexes */
    ncomps = scanptr->comps_in_scan;
    if (ncomps <= 0 || ncomps > MAX_COMPS_IN_SCAN)
        ERREXIT2(cinfo, JERR_COMPONENT_COUNT, ncomps, MAX_COMPS_IN_SCAN);
    for (ci = 0; ci < ncomps; ci++) {
        thisi = scanptr->component_index[ci];
        if (thisi < 0 || thisi >= cinfo->num_components)
            ERREXIT1(cinfo, JERR_BAD_SCAN_SCRIPT, scanno);
        /* Components must appear in SOF order within each scan */
        if (ci > 0 && thisi <= scanptr->component_index[ci-1])
            ERREXIT1(cinfo, JERR_BAD_SCAN_SCRIPT, scanno);
    }
    /* Validate progression parameters */
    Ss = scanptr->Ss;
    Se = scanptr->Se;
    Ah = scanptr->Ah;
    Al = scanptr->Al;
    if (cinfo->progressive_mode) {
#ifdef C_PROGRESSIVE_SUPPORTED
        /* The JPEG spec simply gives the ranges 0..13 for Ah and Al, but that
         * seems wrong: the upper bound ought to depend on data precision.
         * Perhaps they really meant 0..N+1 for N-bit precision.
         * Here we allow 0..10 for 8-bit data; Al larger than 10 results in
         * out-of-range reconstructed DC values during the first DC scan,
         * which might cause problems for some decoders.
         */
        #if BITS_IN_JSAMPLE == 8
        #define MAX_AH_AL 10
        #else
        #define MAX_AH_AL 13
        #endif
        if (Ss < 0 || Ss >= DCTSIZE2 || Se < Ss || Se >= DCTSIZE2 ||
            Ah < 0 || Ah > MAX_AH_AL || Al < 0 || Al > MAX_AH_AL)
            ERREXIT1(cinfo, JERR_BAD_PROG_SCRIPT, scanno);
        if (Ss == 0) {
            if (Se != 0) /* DC and AC together not OK */
                ERREXIT1(cinfo, JERR_BAD_PROG_SCRIPT, scanno);
        } else {
            if (ncomps != 1) /* AC scans must be for only one component */
                ERREXIT1(cinfo, JERR_BAD_PROG_SCRIPT, scanno);
        }
        for (ci = 0; ci < ncomps; ci++) {
            last_bitpos_ptr = & last_bitpos[scanptr->component_index[ci]][0];
            if (Ss != 0 && last_bitpos_ptr[0] < 0) /* AC without prior DC scan */
                ERREXIT1(cinfo, JERR_BAD_PROG_SCRIPT, scanno);
            for (coefi = Ss; coefi <= Se; coefi++) {
                if (last_bitpos_ptr[coefi] < 0) {
                    /* first scan of this coefficient */
                    if (Ah != 0)
                        ERREXIT1(cinfo, JERR_BAD_PROG_SCRIPT, scanno);
                } else {
                    /* not first scan */
                    if (Ah != last_bitpos_ptr[coefi] || Al != Ah-1)
                        ERREXIT1(cinfo, JERR_BAD_PROG_SCRIPT, scanno);
                }
                last_bitpos_ptr[coefi] = Al;
            }
        }
    }
}

#endif
} else {
    /* For sequential JPEG, all progression parameters must be these: */
    if (Ss != 0 || Se != DCTSIZE2-1 || Ah != 0 || Al != 0)
        ERREXIT1(cinfo, JERR_BAD_PROG_SCRIPT, scanno);
    /* Make sure components are not sent twice */
    for (ci = 0; ci < ncomps; ci++) {
        thisi = scanptr->component_index[ci];
        if (component_sent[thisi])
            ERREXIT1(cinfo, JERR_BAD_SCAN_SCRIPT, scanno);
        component_sent[thisi] = TRUE;
    }
}
}

```

```

/* Now verify that everything got sent. */
if (cinfo->progressive_mode) {
#ifdef C_PROGRESSIVE_SUPPORTED
/* For progressive mode, we only check that at least some DC data
 * got sent for each component; the spec does not require that all bits
 * of all coefficients be transmitted. Would it be wiser to enforce
 * transmission of all coefficient bits??
 */
for (ci = 0; ci < cinfo->num_components; ci++) {
    if (last_bitpos[ci][0] < 0)
        ERREXIT(cinfo, JERR_MISSING_DATA);
}
#endif
} else {
for (ci = 0; ci < cinfo->num_components; ci++) {
    if (! component_sent[ci])
        ERREXIT(cinfo, JERR_MISSING_DATA);
}
}
}

#endif /* C_MULTISCAN_FILES_SUPPORTED */

```

```

LOCAL(void)
select_scan_parameters (j_compress_ptr cinfo)
/* Set up the scan parameters for the current scan */
{
    int ci;

#ifdef C_MULTISCAN_FILES_SUPPORTED
    if (cinfo->scan_info != NULL) {
        /* Prepare for current scan --- the script is already validated */
        my_master_ptr master = (my_master_ptr) cinfo->master;
        const jpeg_scan_info * scanptr = cinfo->scan_info + master->scan_number;

        cinfo->comps_in_scan = scanptr->comps_in_scan;
        for (ci = 0; ci < scanptr->comps_in_scan; ci++) {
            cinfo->cur_comp_info[ci] =
                &cinfo->comp_info[scanptr->component_index[ci]];
        }
        cinfo->Ss = scanptr->Ss;
        cinfo->Se = scanptr->Se;
        cinfo->Ah = scanptr->Ah;
        cinfo->Al = scanptr->Al;
    }
    else
#endif
    {
        /* Prepare for single sequential-JPEG scan containing all components */
        if (cinfo->num_components > MAX_COMPS_IN_SCAN)
            ERREXIT2(cinfo, JERR_COMPONENT_COUNT, cinfo->num_components,
                MAX_COMPS_IN_SCAN);
        cinfo->comps_in_scan = cinfo->num_components;
        for (ci = 0; ci < cinfo->num_components; ci++) {
            cinfo->cur_comp_info[ci] = &cinfo->comp_info[ci];
        }
        cinfo->Ss = 0;
        cinfo->Se = DCTSIZE2-1;
        cinfo->Ah = 0;
        cinfo->Al = 0;
    }
}

```

```

LOCAL(void)
per_scan_setup (j_compress_ptr cinfo)
/* Do computations that are needed before processing a JPEG scan */
/* cinfo->comps_in_scan and cinfo->cur_comp_info[] are already set */
{
    int ci, mcublk, tmp;
    jpeg_component_info *comp_ptr;

    if (cinfo->comps_in_scan == 1) {
        /* Noninterleaved (single-component) scan */
        comp_ptr = cinfo->cur_comp_info[0];

        /* Overall image size in MCUs */

```

```

cinfo->MCUs_per_row = compptr->width_in_blocks;
cinfo->MCU_rows_in_scan = compptr->height_in_blocks;

/* For noninterleaved scan, always one block per MCU */
compptr->MCU_width = 1;
compptr->MCU_height = 1;
compptr->MCU_blocks = 1;
compptr->MCU_sample_width = DCTSIZE;
compptr->last_col_width = 1;
/* For noninterleaved scans, it is convenient to define last_row_height
 * as the number of block rows present in the last iMCU row.
 */
tmp = (int) (compptr->height_in_blocks % compptr->v_samp_factor);
if (tmp == 0) tmp = compptr->v_samp_factor;
compptr->last_row_height = tmp;

/* Prepare array describing MCU composition */
cinfo->blocks_in_MCU = 1;
cinfo->MCU_membership[0] = 0;

} else {

/* Interleaved (multi-component) scan */
if (cinfo->comps_in_scan <= 0 || cinfo->comps_in_scan > MAX_COMPS_IN_SCAN)
    ERREXIT2(cinfo, JERR_COMPONENT_COUNT, cinfo->comps_in_scan,
             MAX_COMPS_IN_SCAN);

/* Overall image size in MCUs */
cinfo->MCUs_per_row = (JDIMENSION)
    jdiv_round_up((long) cinfo->image_width,
                  (long) (cinfo->max_h_samp_factor*DCTSIZE));
cinfo->MCU_rows_in_scan = (JDIMENSION)
    jdiv_round_up((long) cinfo->image_height,
                  (long) (cinfo->max_v_samp_factor*DCTSIZE));

cinfo->blocks_in_MCU = 0;

for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
    compptr = cinfo->cur_comp_info[ci];
    /* Sampling factors give # of blocks of component in each MCU */
    compptr->MCU_width = compptr->h_samp_factor;
    compptr->MCU_height = compptr->v_samp_factor;
    compptr->MCU_blocks = compptr->MCU_width * compptr->MCU_height;
    compptr->MCU_sample_width = compptr->MCU_width * DCTSIZE;
    /* Figure number of non-dummy blocks in last MCU column & row */
    tmp = (int) (compptr->width_in_blocks % compptr->MCU_width);
    if (tmp == 0) tmp = compptr->MCU_width;
    compptr->last_col_width = tmp;
    tmp = (int) (compptr->height_in_blocks % compptr->MCU_height);
    if (tmp == 0) tmp = compptr->MCU_height;
    compptr->last_row_height = tmp;
    /* Prepare array describing MCU composition */
    mcublk = compptr->MCU_blocks;
    if (cinfo->blocks_in_MCU + mcublk > C_MAX_BLOCKS_IN_MCU)
        ERREXIT(cinfo, JERR_BAD_MCU_SIZE);
    while (mcublk-- > 0) {
        cinfo->MCU_membership[cinfo->blocks_in_MCU++] = ci;
    }
}

/* Convert restart specified in rows to actual MCU count. */
/* Note that count must fit in 16 bits, so we provide limiting. */
if (cinfo->restart_in_rows > 0) {
    long nominal = (long) cinfo->restart_in_rows * (long) cinfo->MCUs_per_row;
    cinfo->restart_interval = (unsigned int) MIN(nominal, 65535L);
}

/*
 * Per-pass setup.
 * This is called at the beginning of each pass. We determine which modules
 * will be active during this pass and give them appropriate start_pass calls.
 * We also set is_last_pass to indicate whether any more passes will be
 * required.
 */

```

```

METHODDEF(void)

```

```

prepare_for_pass (j_compress_ptr cinfo)
{
    my_master_ptr master = (my_master_ptr) cinfo->master;

    switch (master->pass_type) {
    case main_pass:
        /* Initial pass: will collect input data, and do either Huffman
         * optimization or data output for the first scan.
         */
        select_scan_parameters(cinfo);
        per_scan_setup(cinfo);
        if (! cinfo->raw_data_in) {
            (*cinfo->cconvert->start_pass) (cinfo);
            (*cinfo->downsample->start_pass) (cinfo);
            (*cinfo->prep->start_pass) (cinfo, JBUF_PASS_THRU);
        }
        (*cinfo->fdct->start_pass) (cinfo);
        (*cinfo->entropy->start_pass) (cinfo, cinfo->optimize_coding);
        (*cinfo->coef->start_pass) (cinfo,
            (master->total_passes > 1 ?
             JBUF_SAVE_AND_PASS : JBUF_PASS_THRU));
        (*cinfo->main->start_pass) (cinfo, JBUF_PASS_THRU);
        if (cinfo->optimize_coding) {
            /* No immediate data output; postpone writing frame/scan headers */
            master->pub.call_pass_startup = FALSE;
        } else {
            /* Will write frame/scan headers at first jpeg_write_scanlines call */
            master->pub.call_pass_startup = TRUE;
        }
        break;
#ifdef ENTROPY_OPT_SUPPORTED
    case huff_opt_pass:
        /* Do Huffman optimization for a scan after the first one. */
        select_scan_parameters(cinfo);
        per_scan_setup(cinfo);
        if (cinfo->Ss != 0 || cinfo->Ah == 0 || cinfo->arith_code) {
            (*cinfo->entropy->start_pass) (cinfo, TRUE);
            (*cinfo->coef->start_pass) (cinfo, JBUF_CRANK_DEST);
            master->pub.call_pass_startup = FALSE;
            break;
        }
        /* Special case: Huffman DC refinement scans need no Huffman table
         * and therefore we can skip the optimization pass for them.
         */
        master->pass_type = output_pass;
        master->pass_number++;
        /* FALLTHROUGH */
    case output_pass:
        /* Do a data-output pass. */
        /* We need not repeat per-scan setup if prior optimization pass did it. */
        if (! cinfo->optimize_coding) {
            select_scan_parameters(cinfo);
            per_scan_setup(cinfo);
        }
        (*cinfo->entropy->start_pass) (cinfo, FALSE);
        (*cinfo->coef->start_pass) (cinfo, JBUF_CRANK_DEST);
        /* We emit frame/scan headers now */
        if (master->scan_number == 0)
            (*cinfo->marker->write_frame_header) (cinfo);
        (*cinfo->marker->write_scan_header) (cinfo);
        master->pub.call_pass_startup = FALSE;
        break;
    default:
        ERREXIT(cinfo, JERR_NOT_COMPILED);
    }

    master->pub.is_last_pass = (master->pass_number == master->total_passes-1);

    /* Set up progress monitor's pass info if present */
    if (cinfo->progress != NULL) {
        cinfo->progress->completed_passes = master->pass_number;
        cinfo->progress->total_passes = master->total_passes;
    }
}

/*
 * Special start-of-pass hook.
 * This is called by jpeg_write_scanlines if call_pass_startup is TRUE.
 */

```

```

* In single-pass processing, we need this hook because we don't want to
* write frame/scan headers during jpeg_start_compress; we want to let the
* application write COM markers etc. between jpeg_start_compress and the
* jpeg_write_scanlines loop.
* In multi-pass processing, this routine is not used.
*/

```

```

METHODDEF(void)
pass_startup (j_compress_ptr cinfo)
{
    cinfo->master->call_pass_startup = FALSE; /* reset flag so call only once */

    (*cinfo->marker->write_frame_header) (cinfo);
    (*cinfo->marker->write_scan_header) (cinfo);
}

/*
 * Finish up at end of pass.
 */

```

```

METHODDEF(void)
finish_pass_master (j_compress_ptr cinfo)
{
    my_master_ptr master = (my_master_ptr) cinfo->master;

    /* The entropy coder always needs an end-of-pass call,
     * either to analyze statistics or to flush its output buffer.
     */
    (*cinfo->entropy->finish_pass) (cinfo);

    /* Update state for next pass */
    switch (master->pass_type) {
case main_pass:
    /* next pass is either output of scan 0 (after optimization)
     * or output of scan 1 (if no optimization).
     */
    master->pass_type = output_pass;
    if (! cinfo->optimize_coding)
        master->scan_number++;
    break;
case huff_opt_pass:
    /* next pass is always output of current scan */
    master->pass_type = output_pass;
    break;
case output_pass:
    /* next pass is either optimization or output of next scan */
    if (cinfo->optimize_coding)
        master->pass_type = huff_opt_pass;
    master->scan_number++;
    break;
}
    master->pass_number++;
}

/*
 * Initialize master compression control.
 */

```

```

GLOBAL(void)
jinit_c_master_control (j_compress_ptr cinfo, boolean transcode_only)
{
    my_master_ptr master;

    master = (my_master_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            sizeof(my_comp_master));
    cinfo->master = (struct jpeg_comp_master *) master;
    master->pub.prepare_for_pass = prepare_for_pass;
    master->pub.pass_startup = pass_startup;
    master->pub.finish_pass = finish_pass_master;
    master->pub.is_last_pass = FALSE;

    /* Validate parameters, determine derived values */
    initial_setup(cinfo);

    if (cinfo->scan_info != NULL) {
#ifdef C_MULTISCAN_FILES_SUPPORTED

```



```

        validate_script(cinfo);
    #else
        ERREXIT(cinfo, JERR_NOT_COMPILED);
    #endif
    } else {
        cinfo->progressive_mode = FALSE;
        cinfo->num_scans = 1;
    }

    if (cinfo->progressive_mode) /* TEMPORARY HACK ??? */
        cinfo->optimize_coding = TRUE; /* assume default tables no good for progressive mode */

    /* Initialize my private state */
    if (transcode_only) {
        /* no main pass in transcoding */
        if (cinfo->optimize_coding)
            master->pass_type = huff_opt_pass;
        else
            master->pass_type = output_pass;
    } else {
        /* for normal compression, first pass is always this type: */
        master->pass_type = main_pass;
    }
    master->scan_number = 0;
    master->pass_number = 0;
    if (cinfo->optimize_coding)
        master->total_passes = cinfo->num_scans * 2;
    else
        master->total_passes = cinfo->num_scans;
}

```

```

/*
 * jcomapi.c
 *
 * Copyright (C) 1994-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains application interface routines that are used for both
 * compression and decompression.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/*
 * Abort processing of a JPEG compression or decompression operation,
 * but don't destroy the object itself.
 *
 * For this, we merely clean up all the nonpermanent memory pools.
 * Note that temp files (virtual arrays) are not allowed to belong to
 * the permanent pool, so we will be able to close all temp files here.
 * Closing a data source or destination, if necessary, is the application's
 * responsibility.
 */

GLOBAL(void)
jpeg_abort (j_common_ptr cinfo)
{
    int pool;

    /* Do nothing if called on a not-initialized or destroyed JPEG object. */
    if (cinfo->mem == NULL)
        return;

    /* Releasing pools in reverse order might help avoid fragmentation
     * with some (brain-damaged) malloc libraries.
     */
    for (pool = JPOOL_NUMPOOLS-1; pool > JPOOL_PERMANENT; pool--) {
        (*cinfo->mem->free_pool) (cinfo, pool);
    }

    /* Reset overall state for possible reuse of object */
    if (cinfo->is_decompressor) {
        cinfo->global_state = DSTATE_START;
        /* Try to keep application from accessing now-deleted marker list.
         * A bit kludgy to do it here, but this is the most central place.
         */
        ((j_decompress_ptr) cinfo)->marker_list = NULL;
    } else {
        cinfo->global_state = CSTATE_START;
    }
}

/*
 * Destruction of a JPEG object.
 *
 * Everything gets deallocated except the master jpeg_compress_struct itself
 * and the error manager struct.  Both of these are supplied by the application
 * and must be freed, if necessary, by the application.  (Often they are on
 * the stack and so don't need to be freed anyway.)
 * Closing a data source or destination, if necessary, is the application's
 * responsibility.
 */

GLOBAL(void)
jpeg_destroy (j_common_ptr cinfo)
{
    /* We need only tell the memory manager to release everything. */
    /* NB: mem pointer is NULL if memory mgr failed to initialize. */
    if (cinfo->mem != NULL)
        (*cinfo->mem->self_destruct) (cinfo);
    cinfo->mem = NULL; /* be safe if jpeg_destroy is called twice */
    cinfo->global_state = 0; /* mark it destroyed */
}

/*

```



```

/*
 * jcpparam.c
 *
 * Copyright (C) 1991-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains optional default-setting code for the JPEG compressor.
 * Applications do not have to use this file, but those that don't use it
 * must know a lot more about the innards of the JPEG code.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/*
 * Quantization table setup routines
 */

GLOBAL(void)
jpeg_add_quant_table (j_compress_ptr cinfo, int which_tbl,
                     const unsigned int *basic_table,
                     int scale_factor, boolean force_baseline)
/* Define a quantization table equal to the basic_table times
 * a scale factor (given as a percentage).
 * If force_baseline is TRUE, the computed quantization table entries
 * are limited to 1..255 for JPEG baseline compatibility.
 */
{
  JQUANT_TBL ** qtblptr;
  int i;
  long temp;

  /* Safety check to ensure start_compress not called yet. */
  if (cinfo->global_state != CSTATE_START)
    ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);

  if (which_tbl < 0 || which_tbl >= NUM_QUANT_TBLS)
    ERREXIT1(cinfo, JERR_DQT_INDEX, which_tbl);

  qtblptr = & cinfo->quant_tbl_ptrs[which_tbl];

  if (*qtblptr == NULL)
    *qtblptr = jpeg_alloc_quant_table((j_common_ptr) cinfo);

  for (i = 0; i < DCTSIZE2; i++) {
    temp = ((long) basic_table[i] * scale_factor + 50L) / 100L;
    /* limit the values to the valid range */
    if (temp <= 0L) temp = 1L;
    if (temp > 32767L) temp = 32767L; /* max quantizer needed for 12 bits */
    if (force_baseline && temp > 255L)
      temp = 255L; /* limit to baseline range if requested */
    (*qtblptr)->quantval[i] = (UINT16) temp;
  }

  /* Initialize sent_table FALSE so table will be written to JPEG file. */
  (*qtblptr)->sent_table = FALSE;
}

GLOBAL(void)
jpeg_set_linear_quality (j_compress_ptr cinfo, int scale_factor,
                        boolean force_baseline)
/* Set or change the 'quality' (quantization) setting, using default tables
 * and a straight percentage-scaling quality scale. In most cases it's better
 * to use jpeg_set_quality (below); this entry point is provided for
 * applications that insist on a linear percentage scaling.
 */
{
  /* These are the sample quantization tables given in JPEG spec section K.1.
   * The spec says that the values given produce "good" quality, and
   * when divided by 2, "very good" quality.
   */
  static const unsigned int std_luminance_quant_tbl[DCTSIZE2] = {
    16, 11, 10, 16, 24, 40, 51, 61,
    12, 12, 14, 19, 26, 58, 60, 55,
    14, 13, 16, 24, 40, 57, 69, 56,
    14, 17, 22, 29, 51, 87, 80, 62,
  }

```

```

    18, 22, 37, 56, 68, 109, 103, 77,
    24, 35, 55, 64, 81, 104, 113, 92,
    49, 64, 78, 87, 103, 121, 120, 101,
    72, 92, 95, 98, 112, 100, 103, 99
};

static const unsigned int std_chrominance_quant_tbl[DCTSIZE2] = {
    17, 18, 24, 47, 99, 99, 99, 99,
    18, 21, 26, 66, 99, 99, 99, 99,
    24, 26, 56, 99, 99, 99, 99, 99,
    47, 66, 99, 99, 99, 99, 99, 99,
    99, 99, 99, 99, 99, 99, 99, 99,
    99, 99, 99, 99, 99, 99, 99, 99,
    99, 99, 99, 99, 99, 99, 99, 99,
    99, 99, 99, 99, 99, 99, 99, 99
};

/* Set up two quantization tables using the specified scaling */
jpeg_add_quant_table(cinfo, 0, std_luminance_quant_tbl,
    scale_factor, force_baseline);
jpeg_add_quant_table(cinfo, 1, std_chrominance_quant_tbl,
    scale_factor, force_baseline);
}

GLOBAL(int)
jpeg_quality_scaling (int quality)
/* Convert a user-specified quality rating to a percentage scaling factor
 * for an underlying quantization table, using our recommended scaling curve.
 * The input 'quality' factor should be 0 (terrible) to 100 (very good).
 */
{
    /* Safety limit on quality factor. Convert 0 to 1 to avoid zero divide. */
    if (quality <= 0) quality = 1;
    if (quality > 100) quality = 100;

    /* The basic table is used as-is (scaling 100) for a quality of 50.
     * Qualities 50..100 are converted to scaling percentage 200 - 2*Q;
     * note that at Q=100 the scaling is 0, which will cause jpeg_add_quant_table
     * to make all the table entries 1 (hence, minimum quantization loss).
     * Qualities 1..50 are converted to scaling percentage 5000/Q.
     */
    if (quality < 50)
        quality = 5000 / quality;
    else
        quality = 200 - quality*2;
    return quality;
}

GLOBAL(void)
jpeg_set_quality (j_compress_ptr cinfo, int quality, boolean force_baseline)
/* Set or change the 'quality' (quantization) setting, using default tables.
 * This is the standard quality-adjusting entry point for typical user
 * interfaces; only those who want detailed control over quantization tables
 * would use the preceding three routines directly.
 */
{
    /* Convert user 0-100 rating to percentage scaling */
    quality = jpeg_quality_scaling(quality);

    /* Set up standard quality tables */
    jpeg_set_linear_quality(cinfo, quality, force_baseline);
}

/*
 * Huffman table setup routines
 */

LOCAL(void)
add_huff_table (j_compress_ptr cinfo,
    JHUFF_TBL **htblptr, const UINT8 *bits, const UINT8 *val)
/* Define a Huffman table */
{
    int nsymbols, len;

    if (*htblptr == NULL)
        *htblptr = jpeg_alloc_huff_table((j_common_ptr) cinfo);
}

```

```

/* Copy the number-of-symbols-of-each-code-length counts */
MEMCOPY((*htblptr)->bits, bits, SIZEOF((*htblptr)->bits));

/* Validate the counts. We do this here mainly so we can copy the right
 * number of symbols from the val[] array, without risking marching off
 * the end of memory. jchuff.c will do a more thorough test later.
 */
nsymbols = 0;
for (len = 1; len <= 16; len++)
    nsymbols += bits[len];
if (nsymbols < 1 || nsymbols > 256)
    ERREXIT(cinfo, JERR_BAD_HUFF_TABLE);

MEMCOPY((*htblptr)->huffval, val, nsymbols * SIZEOF(UINT8));

/* Initialize sent_table FALSE so table will be written to JPEG file. */
(*htblptr)->sent_table = FALSE;
}

```

```

LOCAL(void)
std_huff_tables (j_compress_ptr cinfo)
/* Set up the standard Huffman tables (cf. JPEG standard section K.3) */
/* IMPORTANT: these are only valid for 8-bit data precision! */
{
    static const UINT8 bits_dc_luminance[17] =
        { /* 0-base */ 0, 0, 1, 5, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 };
    static const UINT8 val_dc_luminance[] =
        { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };

    static const UINT8 bits_dc_chrominance[17] =
        { /* 0-base */ 0, 0, 3, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 };
    static const UINT8 val_dc_chrominance[] =
        { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };

    static const UINT8 bits_ac_luminance[17] =
        { /* 0-base */ 0, 0, 2, 1, 3, 3, 2, 4, 3, 5, 5, 4, 4, 0, 0, 1, 0x7d };
    static const UINT8 val_ac_luminance[] =
        { 0x01, 0x02, 0x03, 0x00, 0x04, 0x11, 0x05, 0x12,
          0x21, 0x31, 0x41, 0x06, 0x13, 0x51, 0x61, 0x07,
          0x22, 0x71, 0x14, 0x32, 0x81, 0x91, 0xa1, 0x08,
          0x23, 0x42, 0xb1, 0xc1, 0x15, 0x52, 0xd1, 0xf0,
          0x24, 0x33, 0x62, 0x72, 0x82, 0x09, 0x0a, 0x16,
          0x17, 0x18, 0x19, 0x1a, 0x25, 0x26, 0x27, 0x28,
          0x29, 0x2a, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
          0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49,
          0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59,
          0x5a, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69,
          0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79,
          0x7a, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89,
          0x8a, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98,
          0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
          0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6,
          0xb7, 0xb8, 0xb9, 0xba, 0xc2, 0xc3, 0xc4, 0xc5,
          0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2, 0xd3, 0xd4,
          0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xe1, 0xe2,
          0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9, 0xea,
          0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8,
          0xf9, 0xfa };

    static const UINT8 bits_ac_chrominance[17] =
        { /* 0-base */ 0, 0, 2, 1, 2, 4, 4, 3, 4, 7, 5, 4, 4, 0, 1, 2, 0x77 };
    static const UINT8 val_ac_chrominance[] =
        { 0x00, 0x01, 0x02, 0x03, 0x11, 0x04, 0x05, 0x21,
          0x31, 0x06, 0x12, 0x41, 0x51, 0x07, 0x61, 0x71,
          0x13, 0x22, 0x32, 0x81, 0x08, 0x14, 0x42, 0x91,
          0xa1, 0xb1, 0xc1, 0x09, 0x23, 0x33, 0x52, 0xf0,
          0x15, 0x62, 0x72, 0xd1, 0x0a, 0x16, 0x24, 0x34,
          0xe1, 0x25, 0xf1, 0x17, 0x18, 0x19, 0x1a, 0x26,
          0x27, 0x28, 0x29, 0x2a, 0x35, 0x36, 0x37, 0x38,
          0x39, 0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48,
          0x49, 0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58,
          0x59, 0x5a, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68,
          0x69, 0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78,
          0x79, 0x7a, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
          0x88, 0x89, 0x8a, 0x92, 0x93, 0x94, 0x95, 0x96,
          0x97, 0x98, 0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5,
          0xa6, 0xa7, 0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4,
          0xb5, 0xb6, 0xb7, 0xb8, 0xb9, 0xba, 0xc2, 0xc3,
          0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2,

```

```

    0xd3, 0xd4, 0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda,
    0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9,
    0xea, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8,
    0xf9, 0xfa };

add_huff_table(cinfo, &cinfo->dc_huff_tbl_ptrs[0],
    bits_dc_luminance, val_dc_luminance);
add_huff_table(cinfo, &cinfo->ac_huff_tbl_ptrs[0],
    bits_ac_luminance, val_ac_luminance);
add_huff_table(cinfo, &cinfo->dc_huff_tbl_ptrs[1],
    bits_dc_chrominance, val_dc_chrominance);
add_huff_table(cinfo, &cinfo->ac_huff_tbl_ptrs[1],
    bits_ac_chrominance, val_ac_chrominance);
}

/*
 * Default parameter setup for compression.
 *
 * Applications that don't choose to use this routine must do their
 * own setup of all these parameters. Alternately, you can call this
 * to establish defaults and then alter parameters selectively. This
 * is the recommended approach since, if we add any new parameters,
 * your code will still work (they'll be set to reasonable defaults).
 */

GLOBAL(void)
jpeg_set_defaults (j_compress_ptr cinfo)
{
    int i;

    /* Safety check to ensure start_compress not called yet. */
    if (cinfo->global_state != CSTATE_START)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);

    /* Allocate comp_info array large enough for maximum component count.
     * Array is made permanent in case application wants to compress
     * multiple images at same param settings.
     */
    if (cinfo->comp_info == NULL)
        cinfo->comp_info = (jpeg_component_info *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_PERMANENT,
                MAX_COMPONENTS * sizeof(jpeg_component_info));

    /* Initialize everything not dependent on the color space */
    cinfo->data_precision = BITS_IN_JSAMPLE;
    /* Set up two quantization tables using default quality of 75 */
    jpeg_set_quality(cinfo, 75, TRUE);
    /* Set up two Huffman tables */
    std_huff_tables(cinfo);

    /* Initialize default arithmetic coding conditioning */
    for (i = 0; i < NUM_ARITH_TBLS; i++) {
        cinfo->arith_dc_L[i] = 0;
        cinfo->arith_dc_U[i] = 1;
        cinfo->arith_ac_K[i] = 5;
    }

    /* Default is no multiple-scan output */
    cinfo->scan_info = NULL;
    cinfo->num_scans = 0;

    /* Expect normal source image, not raw downsampled data */
    cinfo->raw_data_in = FALSE;

    /* Use Huffman coding, not arithmetic coding, by default */
    cinfo->arith_code = FALSE;

    /* By default, don't do extra passes to optimize entropy coding */
    cinfo->optimize_coding = FALSE;
    /* The standard Huffman tables are only valid for 8-bit data precision.
     * If the precision is higher, force optimization on so that usable
     * tables will be computed. This test can be removed if default tables
     * are supplied that are valid for the desired precision.
     */
    if (cinfo->data_precision > 8)
        cinfo->optimize_coding = TRUE;

    /* By default, use the simpler non-cosited sampling alignment */

```

```

cinfo->CCIR601_sampling = FALSE;

/* No input smoothing */
cinfo->smoothing_factor = 0;

/* DCT algorithm preference */
cinfo->dct_method = JDCT_DEFAULT;

/* No restart markers */
cinfo->restart_interval = 0;
cinfo->restart_in_rows = 0;

/* Fill in default JFIF marker parameters. Note that whether the marker
 * will actually be written is determined by jpeg_set_colorspace.
 */
/* By default, the library emits JFIF version code 1.01.
 * An application that wants to emit JFIF 1.02 extension markers should set
 * JFIF_minor_version to 2. We could probably get away with just defaulting
 * to 1.02, but there may still be some decoders in use that will complain
 * about that; saying 1.01 should minimize compatibility problems.
 */
cinfo->JFIF_major_version = 1; /* Default JFIF version = 1.01 */
cinfo->JFIF_minor_version = 1;
cinfo->density_unit = 0; /* Pixel size is unknown by default */
cinfo->X_density = 1; /* Pixel aspect ratio is square by default */
cinfo->Y_density = 1;

/* Choose JPEG colorspace based on input space, set defaults accordingly */
jpeg_default_colorspace(cinfo);
}

/*
 * Select an appropriate JPEG colorspace for in_color_space.
 */
GLOBAL(void)
jpeg_default_colorspace (j_compress_ptr cinfo)
{
    switch (cinfo->in_color_space) {
        case JCS_GRAYSCALE:
            jpeg_set_colorspace(cinfo, JCS_GRAYSCALE);
            break;
        case JCS_RGB:
            jpeg_set_colorspace(cinfo, JCS_YCbCr);
            break;
        case JCS_YCbCr:
            jpeg_set_colorspace(cinfo, JCS_YCbCr);
            break;
        case JCS_CMYK:
            jpeg_set_colorspace(cinfo, JCS_CMYK); /* By default, no translation */
            break;
        case JCS_YCCK:
            jpeg_set_colorspace(cinfo, JCS_YCCK);
            break;
        case JCS_UNKNOWN:
            jpeg_set_colorspace(cinfo, JCS_UNKNOWN);
            break;
        default:
            ERREXIT(cinfo, JERR_BAD_IN_COLORSPACE);
    }
}

/*
 * Set the JPEG colorspace, and choose colorspace-dependent default values.
 */
GLOBAL(void)
jpeg_set_colorspace (j_compress_ptr cinfo, J_COLOR_SPACE colorspace)
{
    jpeg_component_info * comp_ptr;
    int ci;

#define SET_COMP(index,id,hsamp,vsamp,quant,dctbl,actbl) \
    {comp_ptr = &cinfo->comp_info[index], \
    comp_ptr->component_id = (id), \
    comp_ptr->h_samp_factor = (hsamp), \
    comp_ptr->v_samp_factor = (vsamp), \
}

```



```

comp_ptr->quant_tbl_no = (quant), \
comp_ptr->dc_tbl_no = (dctbl), \
comp_ptr->ac_tbl_no = (actbl) )

/* Safety check to ensure start_compress not called yet. */
if (cinfo->global_state != CSTATE_START)
    ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);

/* For all colorspace, we use Q and Huff tables 0 for luminance components,
 * tables 1 for chrominance components.
 */

cinfo->jpeg_color_space = colorspace;

cinfo->write_JFIF_header = FALSE; /* No marker for non-JFIF colorspace */
cinfo->write_Adobe_marker = FALSE; /* write no Adobe marker by default */

switch (cinfo->jpeg_color_space) {
case JCS_GRAYSCALE:
    cinfo->write_JFIF_header = TRUE; /* Write a JFIF marker */
    cinfo->num_components = 1;
    /* JFIF specifies component ID 1 */
    SET_COMP(0, 1, 1, 1, 0, 0, 0);
    break;
case JCS_RGB:
    cinfo->write_Adobe_marker = TRUE; /* write Adobe marker to flag RGB */
    cinfo->num_components = 3;
    SET_COMP(0, 0x52 /* 'R' */, 1, 1, 0, 0, 0);
    SET_COMP(1, 0x47 /* 'G' */, 1, 1, 0, 0, 0);
    SET_COMP(2, 0x42 /* 'B' */, 1, 1, 0, 0, 0);
    break;
case JCS_YCbCr:
    cinfo->write_JFIF_header = TRUE; /* Write a JFIF marker */
    cinfo->num_components = 3;
    /* JFIF specifies component IDs 1,2,3 */
    /* We default to 2x2 subsamples of chrominance */
    SET_COMP(0, 1, 2, 2, 0, 0, 0);
    SET_COMP(1, 2, 1, 1, 1, 1, 1);
    SET_COMP(2, 3, 1, 1, 1, 1, 1);
    break;
case JCS_CMYK:
    cinfo->write_Adobe_marker = TRUE; /* write Adobe marker to flag CMYK */
    cinfo->num_components = 4;
    SET_COMP(0, 0x43 /* 'C' */, 1, 1, 0, 0, 0);
    SET_COMP(1, 0x4D /* 'M' */, 1, 1, 0, 0, 0);
    SET_COMP(2, 0x59 /* 'Y' */, 1, 1, 0, 0, 0);
    SET_COMP(3, 0x4B /* 'K' */, 1, 1, 0, 0, 0);
    break;
case JCS_YCCK:
    cinfo->write_Adobe_marker = TRUE; /* write Adobe marker to flag YCCK */
    cinfo->num_components = 4;
    SET_COMP(0, 1, 2, 2, 0, 0, 0);
    SET_COMP(1, 2, 1, 1, 1, 1, 1);
    SET_COMP(2, 3, 1, 1, 1, 1, 1);
    SET_COMP(3, 4, 2, 2, 0, 0, 0);
    break;
case JCS_UNKNOWN:
    cinfo->num_components = cinfo->input_components;
    if (cinfo->num_components < 1 || cinfo->num_components > MAX_COMPONENTS)
        ERREXIT2(cinfo, JERR_COMPONENT_COUNT, cinfo->num_components,
            MAX_COMPONENTS);
    for (ci = 0; ci < cinfo->num_components; ci++) {
        SET_COMP(ci, ci, 1, 1, 0, 0, 0);
    }
    break;
default:
    ERREXIT(cinfo, JERR_BAD_J_COLORSPACE);
}
}

#ifdef C_PROGRESSIVE_SUPPORTED

LOCAL(jpeg_scan_info *)
fill_a_scan (jpeg_scan_info * scan_ptr, int ci,
             int Ss, int Se, int Ah, int Al)
/* Support routine: generate one scan for specified component */
{
    scan_ptr->comps_in_scan = 1;
    scan_ptr->component_index[0] = ci;
}

```

```

scanptr->Ss = Ss;
scanptr->Se = Se;
scanptr->Ah = Ah;
scanptr->Al = Al;
scanptr++;
return scanptr;
}

LOCAL(jpeg_scan_info *)
fill_scans (jpeg_scan_info * scanptr, int ncomps,
            int Ss, int Se, int Ah, int Al)
/* Support routine: generate one scan for each component */
{
    int ci;

    for (ci = 0; ci < ncomps; ci++) {
        scanptr->comps_in_scan = 1;
        scanptr->component_index[0] = ci;
        scanptr->Ss = Ss;
        scanptr->Se = Se;
        scanptr->Ah = Ah;
        scanptr->Al = Al;
        scanptr++;
    }
    return scanptr;
}

LOCAL(jpeg_scan_info *)
fill_dc_scans (jpeg_scan_info * scanptr, int ncomps, int Ah, int Al)
/* Support routine: generate interleaved DC scan if possible, else N scans */
{
    int ci;

    if (ncomps <= MAX_COMPS_IN_SCAN) {
        /* Single interleaved DC scan */
        scanptr->comps_in_scan = ncomps;
        for (ci = 0; ci < ncomps; ci++)
            scanptr->component_index[ci] = ci;
        scanptr->Ss = scanptr->Se = 0;
        scanptr->Ah = Ah;
        scanptr->Al = Al;
        scanptr++;
    } else {
        /* Noninterleaved DC scan for each component */
        scanptr = fill_scans(scanptr, ncomps, 0, 0, Ah, Al);
    }
    return scanptr;
}

/* Create a recommended progressive-JPEG script.
   cinfo->num_components and cinfo->jpeg_color_space must be correct.
*/

GLOBAL(void)
jpeg_simple_progression (j_compress_ptr cinfo)
{
    int ncomps = cinfo->num_components;
    int nscans;
    jpeg_scan_info * scanptr;

    /* Safety check to ensure start_compress not called yet. */
    if (cinfo->global_state != CSTATE_START)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);

    /* Figure space needed for script. Calculation must match code below! */
    if (ncomps == 3 && cinfo->jpeg_color_space == JCS_YCbCr) {
        /* Custom script for YCbCr color images. */
        nscans = 10;
    } else {
        /* All-purpose script for other color spaces. */
        if (ncomps > MAX_COMPS_IN_SCAN)
            nscans = 6 * ncomps; /* 2 DC + 4 AC scans per component */
        else
            nscans = 2 + 4 * ncomps; /* 2 DC scans; 4 AC scans per component */
    }

    /* Allocate space for script.
       * We need to put it in the permanent pool in case the application performs
    */

```

```

* multiple compressions without changing the settings. To avoid a memory
* leak if jpeg_simple_progression is called repeatedly for the same JPEG
* object, we try to re-use previously allocated space, and we allocate
* enough space to handle YCbCr even if initially asked for grayscale.
*/
if (cinfo->script_space == NULL || cinfo->script_space_size < nscans) {
    cinfo->script_space_size = MAX(nscans, 10);
    cinfo->script_space = (jpeg_scan_info *)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_PERMANENT,
            cinfo->script_space_size * sizeof(jpeg_scan_info));
}
scanptr = cinfo->script_space;
cinfo->scan_info = scanptr;
cinfo->num_scans = nscans;

if (ncomps == 3 && cinfo->jpeg_color_space == JCS_YCbCr) {
    /* Custom script for YCbCr color images. */
    /* Initial DC scan */
    scanptr = fill_dc_scans(scanptr, ncomps, 0, 1);
    /* Initial AC scan: get some luma data out in a hurry */
    scanptr = fill_a_scan(scanptr, 0, 1, 5, 0, 2);
    /* Chroma data is too small to be worth expending many scans on */
    scanptr = fill_a_scan(scanptr, 2, 1, 63, 0, 1);
    scanptr = fill_a_scan(scanptr, 1, 1, 63, 0, 1);
    /* Complete spectral selection for luma AC */
    scanptr = fill_a_scan(scanptr, 0, 6, 63, 0, 2);
    /* Refine next bit of luma AC */
    scanptr = fill_a_scan(scanptr, 0, 1, 63, 2, 1);
    /* Finish DC successive approximation */
    scanptr = fill_dc_scans(scanptr, ncomps, 1, 0);
    /* Finish AC successive approximation */
    scanptr = fill_a_scan(scanptr, 2, 1, 63, 1, 0);
    scanptr = fill_a_scan(scanptr, 1, 1, 63, 1, 0);
    /* Luma bottom bit comes last since it's usually largest scan */
    scanptr = fill_a_scan(scanptr, 0, 1, 63, 1, 0);
    else {
        /* All-purpose script for other color spaces. */
        /* Successive approximation first pass */
        scanptr = fill_dc_scans(scanptr, ncomps, 0, 1);
        scanptr = fill_scans(scanptr, ncomps, 1, 5, 0, 2);
        scanptr = fill_scans(scanptr, ncomps, 6, 63, 0, 2);
        /* Successive approximation second pass */
        scanptr = fill_scans(scanptr, ncomps, 1, 63, 2, 1);
        /* Successive approximation final pass */
        scanptr = fill_dc_scans(scanptr, ncomps, 1, 0);
        scanptr = fill_scans(scanptr, ncomps, 1, 63, 1, 0);
    }
}
#endif /* C_PROGRESSIVE_SUPPORTED */

```

```

/*
 * jcpuff.c
 *
 * Copyright (C) 1995-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains Huffman entropy encoding routines for progressive JPEG.
 *
 * We do not support output suspension in this module, since the library
 * currently does not allow multiple-scan files to be written with output
 * suspension.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jchuff.h" /* Declarations shared with jchuff.c */

#ifdef C_PROGRESSIVE_SUPPORTED

/* Expanded entropy encoder object for progressive Huffman encoding. */

typedef struct {
  struct jpeg_entropy_encoder pub; /* public fields */

  /* Mode flag: TRUE for optimization, FALSE for actual data output */
  boolean gather_statistics;

  /* Bit-level coding status.
   * next_output_byte/free_in_buffer are local copies of cinfo->dest fields.
   */
  JOCTET * next_output_byte; /* => next byte to write in buffer */
  size_t free_in_buffer; /* # of byte spaces remaining in buffer */
  INT32 put_buffer; /* current bit-accumulation buffer */
  int put_bits; /* # of bits now in it */
  j_compress_ptr cinfo; /* link to cinfo (needed for dump_buffer) */

  /* Coding status for DC components */
  int last_dc_val[MAX_COMPS_IN_SCAN]; /* last DC coef for each component */

  /* Coding status for AC components */
  int ac_tbl_no; /* the table number of the single component */
  unsigned int EOBRUN; /* run length of EOBs */
  unsigned int BE; /* # of buffered correction bits before MCU */
  char * bit_buffer; /* buffer for correction bits (1 per char) */
  /* packing correction bits tightly would save some space but cost time... */

  unsigned int restarts_to_go; /* MCUs left in this restart interval */
  int next_restart_num; /* next restart number to write (0-7) */

  /* Pointers to derived tables (these workspaces have image lifespan).
   * Since any one scan codes only DC or only AC, we only need one set
   * of tables, not one for DC and one for AC.
   */
  c_derived_tbl * derived_tbls[NUM_HUFF_TBLS];

  /* Statistics tables for optimization; again, one set is enough */
  long * count_ptrs[NUM_HUFF_TBLS];
} phuff_entropy_encoder;

typedef phuff_entropy_encoder * phuff_entropy_ptr;

/* MAX_CORR_BITS is the number of bits the AC refinement correction-bit
 * buffer can hold. Larger sizes may slightly improve compression, but
 * 1000 is already well into the realm of overkill.
 * The minimum safe size is 64 bits.
 */

#define MAX_CORR_BITS 1000 /* Max # of correction bits I can buffer */

/* IRIGHT_SHIFT is like RIGHT_SHIFT, but works on int rather than INT32.
 * We assume that int right shift is unsigned if INT32 right shift is,
 * which should be safe.
 */

#ifdef RIGHT_SHIFT_IS_UNSIGNED
#define ISHIFT_TEMPS int ishift_temp;
#define IRIGHT_SHIFT(x,shft) \
  ((ishift_temp = (x)) < 0 ? \

```

```

        (ishift_temp >> (shft)) | ((-0) << (16-(shft))) : \
        (ishift_temp >> (shft)))
#else
#define ISHIFT_TEMPS
#define IRIGHT_SHIFT(x,shft)      ((x) >> (shft))
#endif

/* Forward declarations */
METHODDEF(boolean) encode_mcu_DC_first JPP((j_compress_ptr cinfo,
        JBLOCKROW *MCU_data));
METHODDEF(boolean) encode_mcu_AC_first JPP((j_compress_ptr cinfo,
        JBLOCKROW *MCU_data));
METHODDEF(boolean) encode_mcu_DC_refine JPP((j_compress_ptr cinfo,
        JBLOCKROW *MCU_data));
METHODDEF(boolean) encode_mcu_AC_refine JPP((j_compress_ptr cinfo,
        JBLOCKROW *MCU_data));
METHODDEF(void) finish_pass_phuff JPP((j_compress_ptr cinfo));
METHODDEF(void) finish_pass_gather_phuff JPP((j_compress_ptr cinfo));

/*
 * Initialize for a Huffman-compressed scan using progressive JPEG.
 */

METHODDEF(void)
start_pass_phuff (j_compress_ptr cinfo, boolean gather_statistics)
{
    phuff_entropy_ptr entropy = (phuff_entropy_ptr) cinfo->entropy;
    boolean is_DC_band;
    int ci, tbl;
    jpeg_component_info * compptr;

    entropy->cinfo = cinfo;
    entropy->gather_statistics = gather_statistics;

    is_DC_band = (cinfo->Ss == 0);

    /* We assume jcmaster.c already validated the scan parameters. */

    /* Select execution routines */
    if (cinfo->Ah == 0) {
        if (is_DC_band)
            entropy->pub.encode_mcu = encode_mcu_DC_first;
        else
            entropy->pub.encode_mcu = encode_mcu_AC_first;
    } else {
        if (is_DC_band)
            entropy->pub.encode_mcu = encode_mcu_DC_refine;
        else {
            entropy->pub.encode_mcu = encode_mcu_AC_refine;
            /* AC refinement needs a correction bit buffer */
            if (entropy->bit_buffer == NULL)
                entropy->bit_buffer = (char *)
                    (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                        MAX_CORR_BITS * sizeof(char));
        }
    }

    if (gather_statistics)
        entropy->pub.finish_pass = finish_pass_gather_phuff;
    else
        entropy->pub.finish_pass = finish_pass_phuff;

    /* Only DC coefficients may be interleaved, so cinfo->comps_in_scan = 1
     * for AC coefficients.
     */
    for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
        compptr = cinfo->cur_comp_info[ci];
        /* Initialize DC predictions to 0 */
        entropy->last_dc_val[ci] = 0;
        /* Get table index */
        if (is_DC_band) {
            if (cinfo->Ah != 0) /* DC refinement needs no table */
                continue;
            tbl = compptr->dc_tbl_no;
        } else {
            entropy->ac_tbl_no = tbl = compptr->ac_tbl_no;
        }
        if (gather_statistics) {
            /* Check for invalid table index */
            /* (make_c_derived_tbl does this in the other path) */

```

```

    if (tbl < 0 || tbl >= NUM_HUFF_TBLS)
        ERREXIT1(cinfo, JERR_NO_HUFF_TABLE, tbl);
    /* Allocate and zero the statistics tables */
    /* Note that jpeg_gen_optimal_table expects 257 entries in each table! */
    if (entropy->count_ptrs[tbl] == NULL)
        entropy->count_ptrs[tbl] = (long *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                257 * sizeof(long));
    MEMZERO(entropy->count_ptrs[tbl], 257 * sizeof(long));
} else {
    /* Compute derived values for Huffman table */
    /* We may do this more than once for a table, but it's not expensive */
    jpeg_make_c_derived_tbl(cinfo, is_DC_band, tbl,
        & entropy->derived_tbls[tbl]);
}
}

/* Initialize AC stuff */
entropy->EOBRUN = 0;
entropy->BE = 0;

/* Initialize bit buffer to empty */
entropy->put_buffer = 0;
entropy->put_bits = 0;

/* Initialize restart stuff */
entropy->restarts_to_go = cinfo->restart_interval;
entropy->next_restart_num = 0;
}

/* Outputting bytes to the file.
 * NB: these must be called only when actually outputting,
 * that is, entropy->gather_statistics == FALSE.
 */

/* Emit a byte */
#define emit_byte(entropy, val) \
{ *(entropy)->next_output_byte++ = (JOCTET) (val); \
  if (--(entropy)->free_in_buffer == 0) \
    dump_buffer(entropy); }

LOCAL(void)
dump_buffer (phuff_entropy_ptr entropy)
/* Empty the output buffer; we do not support suspension in this module. */
{
    struct jpeg_destination_mgr * dest = entropy->cinfo->dest;

    if (! (*dest->empty_output_buffer) (entropy->cinfo))
        ERREXIT(entropy->cinfo, JERR_CANT_SUSPEND);
    /* After a successful buffer dump, must reset buffer pointers */
    entropy->next_output_byte = dest->next_output_byte;
    entropy->free_in_buffer = dest->free_in_buffer;
}

/* Outputting bits to the file */

/* Only the right 24 bits of put_buffer are used; the valid bits are
 * left-justified in this part. At most 16 bits can be passed to emit_bits
 * in one call, and we never retain more than 7 bits in put_buffer
 * between calls, so 24 bits are sufficient.
 */

INLINE
LOCAL(void)
emit_bits (phuff_entropy_ptr entropy, unsigned int code, int size)
/* Emit some bits, unless we are in gather mode */
{
    /* This routine is heavily used, so it's worth coding tightly. */
    register INT32 put_buffer = (INT32) code;
    register int put_bits = entropy->put_bits;

    /* if size is 0, caller used an invalid Huffman table entry */
    if (size == 0)
        ERREXIT(entropy->cinfo, JERR_HUFF_MISSING_CODE);

    if (entropy->gather_statistics)
        return; /* do nothing if we're only getting stats */
}

```

```

put_buffer &= (((INT32) 1) << size) - 1; /* mask off any extra bits in code */
put_bits += size; /* new number of bits in buffer */
put_buffer <= 24 - put_bits; /* align incoming bits */
put_buffer |= entropy->put_buffer; /* and merge with old buffer contents */

while (put_bits >= 8) {
    int c = (int) ((put_buffer >> 16) & 0xFF);

    emit_byte(entropy, c);
    if (c == 0xFF) { /* need to stuff a zero byte? */
        emit_byte(entropy, 0);
    }
    put_buffer <= 8;
    put_bits -= 8;
}

entropy->put_buffer = put_buffer; /* update variables */
entropy->put_bits = put_bits;
}

```

```

LOCAL(void)
flush_bits (phuff_entropy_ptr entropy)
{
    emit_bits(entropy, 0x7F, 7); /* fill any partial byte with ones */
    entropy->put_buffer = 0; /* and reset bit-buffer to empty */
    entropy->put_bits = 0;
}

```

```

/* Emit (or just count) a Huffman symbol.
 */

```

```

INLINE
LOCAL(void)
emit_symbol (phuff_entropy_ptr entropy, int tbl_no, int symbol)
{
    if (entropy->gather_statistics)
        entropy->count_ptrs[tbl_no][symbol]++;
    else {
        c_derived_tbl * tbl = entropy->derived_tbls[tbl_no];
        emit_bits(entropy, tbl->ehufco[symbol], tbl->ehufsi[symbol]);
    }
}

```

```

/* Emit bits from a correction bit buffer.
 */

```

```

LOCAL(void)
emit_buffered_bits (phuff_entropy_ptr entropy, char * bufstart,
                    unsigned int nbits)
{
    if (entropy->gather_statistics)
        return; /* no real work */

    while (nbits > 0) {
        emit_bits(entropy, (unsigned int) (*bufstart), 1);
        bufstart++;
        nbits--;
    }
}

```

```

/*
 * Emit any pending EOBRUN symbol.
 */

```

```

LOCAL(void)
emit_eobrun (phuff_entropy_ptr entropy)
{
    register int temp, nbits;

    if (entropy->EOBRUN > 0) { /* if there is any pending EOBRUN */

```

```

temp = entropy->EOBRUN;
nbits = 0;
while ((temp >= 1))
    nbits++;
/* safety check: shouldn't happen given limited correction-bit buffer */
if (nbits > 14)
    ERREXIT(entropy->cinfo, JERR_HUFF_MISSING_CODE);

emit_symbol(entropy, entropy->ac_tbl_no, nbits << 4);
if (nbits)
    emit_bits(entropy, entropy->EOBRUN, nbits);

entropy->EOBRUN = 0;

/* Emit any buffered correction bits */
emit_buffered_bits(entropy, entropy->bit_buffer, entropy->BE);
entropy->BE = 0;
}
}

```

```

/*
 * Emit a restart marker & resynchronize predictions.
 */

```

```

LOCAL(void)
emit_restart (phuff_entropy_ptr entropy, int restart_num)
{
    int ci;

```

```

    emit_eobrun(entropy);
    if (! entropy->gather_statistics) {
        flush_bits(entropy);
        emit_byte(entropy, 0xFF);
        emit_byte(entropy, JPEG_RST0 + restart_num);
    }
    if (entropy->cinfo->Ss == 0) {
        /* Re-initialize DC predictions to 0 */
        for (ci = 0; ci < entropy->cinfo->comps_in_scan; ci++)
            entropy->last_dc_val[ci] = 0;
    } else {
        /* Re-initialize all AC-related fields to 0 */
        entropy->EOBRUN = 0;
        entropy->BE = 0;
    }

```

```

/* MCU encoding for DC initial scan (either spectral selection,
 * or first pass of successive approximation).
 */

```

```

METHODDEF(boolean)
encode_mcu_DC_first (j_compress_ptr cinfo, JBLOCKROW *MCU_data)
{
    phuff_entropy_ptr entropy = (phuff_entropy_ptr) cinfo->entropy;
    register int temp, temp2;
    register int nbits;
    int blk, ci;
    int Al = cinfo->Al;
    JBLOCKROW block;
    jpeg_component_info * comp_ptr;
    ISHIFT_TEMPS

    entropy->next_output_byte = cinfo->dest->next_output_byte;
    entropy->free_in_buffer = cinfo->dest->free_in_buffer;

    /* Emit restart marker if needed */
    if (cinfo->restart_interval)
        if (entropy->restarts_to_go == 0)
            emit_restart(entropy, entropy->next_restart_num);

    /* Encode the MCU data blocks */
    for (blk = 0; blk < cinfo->blocks_in_MCU; blk++) {
        block = MCU_data[blk];
        ci = cinfo->MCU_membership[blk];
        comp_ptr = cinfo->cur_comp_info[ci];
    }

```



```

/* Compute the DC value after the required point transform by A1.
 * This is simply an arithmetic right shift.
 */
temp2 = IRIGHT_SHIFT((int) ((*block)[0]), A1);

/* DC differences are figured on the point-transformed values. */
temp = temp2 - entropy->last_dc_val[ci];
entropy->last_dc_val[ci] = temp2;

/* Encode the DC coefficient difference per section G.1.2.1 */
temp2 = temp;
if (temp < 0) {
    temp = -temp;          /* temp is abs value of input */
    /* For a negative input, want temp2 = bitwise complement of abs(input) */
    /* This code assumes we are on a two's complement machine */
    temp2--;
}

/* Find the number of bits needed for the magnitude of the coefficient */
nbits = 0;
while (temp) {
    nbits++;
    temp >>= 1;
}
/* Check for out-of-range coefficient values.
 * Since we're encoding a difference, the range limit is twice as much.
 */
if (nbits > MAX_COEF_BITS+1)
    ERREXIT(cinfo, JERR_BAD_DCT_COEF);

/* Count/emit the Huffman-coded symbol for the number of bits */
emit_symbol(entropy, compptr->dc_tbl_no, nbits);

/* Emit that number of bits of the value, if positive, */
/* or the complement of its magnitude, if negative. */
if (nbits)                /* emit_bits rejects calls with size 0 */
    emit_bits(entropy, (unsigned int) temp2, nbits);

cinfo->dest->next_output_byte = entropy->next_output_byte;
cinfo->dest->free_in_buffer = entropy->free_in_buffer;

/* Update restart-interval state too */
if (cinfo->restart_interval) {
    if (entropy->restarts_to_go == 0) {
        entropy->restarts_to_go = cinfo->restart_interval;
        entropy->next_restart_num++;
        entropy->next_restart_num &= 7;
    }
    entropy->restarts_to_go--;
}

return TRUE;
}

/*
 * MCU encoding for AC initial scan (either spectral selection,
 * or first pass of successive approximation).
 */

METHODDEF(boolean)
encode_mcu_AC_first (j_compress_ptr cinfo, JBLOCKROW *MCU_data)
{
    phuff_entropy_ptr entropy = (phuff_entropy_ptr) cinfo->entropy;
    register int temp, temp2;
    register int nbits;
    register int r, k;
    int Se = cinfo->Se;
    int A1 = cinfo->A1;
    JBLOCKROW block;

    entropy->next_output_byte = cinfo->dest->next_output_byte;
    entropy->free_in_buffer = cinfo->dest->free_in_buffer;

    /* Emit restart marker if needed */
    if (cinfo->restart_interval)
        if (entropy->restarts_to_go == 0)
            emit_restart(entropy, entropy->next_restart_num);

```

```

/* Encode the MCU data block */
block = MCU_data[0];

/* Encode the AC coefficients per section G.1.2.2, fig. G.3 */

r = 0;          /* r = run length of zeros */

for (k = cinfo->Ss; k <= Se; k++) {
    if ((temp = (*block)[jpeg_natural_order[k]]) == 0) {
        r++;
        continue;
    }
    /* We must apply the point transform by A1. For AC coefficients this
     * is an integer division with rounding towards 0. To do this portably
     * in C, we shift after obtaining the absolute value; so the code is
     * interwoven with finding the abs value (temp) and output bits (temp2).
     */
    if (temp < 0) {
        temp = -temp;          /* temp is abs value of input */
        temp >>= A1;          /* apply the point transform */
        /* For a negative coef, want temp2 = bitwise complement of abs(coef) */
        temp2 = ~temp;
    } else {
        temp >>= A1;          /* apply the point transform */
        temp2 = temp;
    }
    /* Watch out for case that nonzero coef is zero after point transform */
    if (temp == 0) {
        r++;
        continue;
    }

    /* Emit any pending EOBRUN */
    if (entropy->EOBRUN > 0)
        emit_eobrun(entropy);
    /* if run length > 15, must emit special run-length-16 codes (0xF0) */
    while (r > 15) {
        emit_symbol(entropy, entropy->ac_tbl_no, 0xF0);
        r -= 16;
    }

    /* Find the number of bits needed for the magnitude of the coefficient */
    nbits = 1;          /* there must be at least one 1 bit */
    while ((temp >>= 1))
        nbits++;
    /* Check for out-of-range coefficient values */
    if (nbits > MAX_COEF_BITS)
        ERREXIT(cinfo, JERR_BAD_DCT_COEF);

    /* Count/emit Huffman symbol for run length / number of bits */
    emit_symbol(entropy, entropy->ac_tbl_no, (r <= 4) + nbits);

    /* Emit that number of bits of the value, if positive, */
    /* or the complement of its magnitude, if negative. */
    emit_bits(entropy, (unsigned int) temp2, nbits);

    r = 0;          /* reset zero run length */
}

if (r > 0) {          /* If there are trailing zeroes, */
    entropy->EOBRUN++; /* count an EOB */
    if (entropy->EOBRUN == 0x7FFF)
        emit_eobrun(entropy); /* force it out to avoid overflow */
}

cinfo->dest->next_output_byte = entropy->next_output_byte;
cinfo->dest->free_in_buffer = entropy->free_in_buffer;

/* Update restart-interval state too */
if (cinfo->restart_interval) {
    if (entropy->restarts_to_go == 0) {
        entropy->restarts_to_go = cinfo->restart_interval;
        entropy->next_restart_num++;
        entropy->next_restart_num &= 7;
    }
    entropy->restarts_to_go--;
}

return TRUE;

```

```

}

/*
 * MCU encoding for DC successive approximation refinement scan.
 * Note: we assume such scans can be multi-component, although the spec
 * is not very clear on the point.
 */

```

```

METHODDEF(boolean)
encode_mcu_DC_refine (j_compress_ptr cinfo, JBLOCKROW *MCU_data)
{
    phuff_entropy_ptr entropy = (phuff_entropy_ptr) cinfo->entropy;
    register int temp;
    int blkcn;
    int Al = cinfo->Al;
    JBLOCKROW block;

    entropy->next_output_byte = cinfo->dest->next_output_byte;
    entropy->free_in_buffer = cinfo->dest->free_in_buffer;

    /* Emit restart marker if needed */
    if (cinfo->restart_interval)
        if (entropy->restarts_to_go == 0)
            emit_restart(entropy, entropy->next_restart_num);

    /* Encode the MCU data blocks */
    for (blkcn = 0; blkcn < cinfo->blocks_in_MCU; blkcn++) {
        block = MCU_data[blkcn];

        /* We simply emit the Al'th bit of the DC coefficient value. */
        temp = (*block)[0];
        emit_bits(entropy, (unsigned int) (temp >> Al), 1);

        cinfo->dest->next_output_byte = entropy->next_output_byte;
        cinfo->dest->free_in_buffer = entropy->free_in_buffer;

        /* Update restart-interval state too */
        if (cinfo->restart_interval) {
            if (entropy->restarts_to_go == 0) {
                entropy->restarts_to_go = cinfo->restart_interval;
                entropy->next_restart_num++;
                entropy->next_restart_num &= 7;
            }
            entropy->restarts_to_go--;
        }

        return TRUE;
    }
}

/*
 * MCU encoding for AC successive approximation refinement scan.
 */

```

```

METHODDEF(boolean)
encode_mcu_AC_refine (j_compress_ptr cinfo, JBLOCKROW *MCU_data)
{
    phuff_entropy_ptr entropy = (phuff_entropy_ptr) cinfo->entropy;
    register int temp;
    register int r, k;
    int EOB;
    char *BR_buffer;
    unsigned int BR;
    int Se = cinfo->Se;
    int Al = cinfo->Al;
    JBLOCKROW block;
    int absvalues[DCTSIZE2];

    entropy->next_output_byte = cinfo->dest->next_output_byte;
    entropy->free_in_buffer = cinfo->dest->free_in_buffer;

    /* Emit restart marker if needed */
    if (cinfo->restart_interval)
        if (entropy->restarts_to_go == 0)
            emit_restart(entropy, entropy->next_restart_num);

    /* Encode the MCU data block */
    block = MCU_data[0];

```

```

/* It is convenient to make a pre-pass to determine the transformed
 * coefficients' absolute values and the EOB position.
 */
EOB = 0;
for (k = cinfo->Ss; k <= Se; k++) {
    temp = (*block)[jpeg_natural_order[k]];
    /* We must apply the point transform by A1. For AC coefficients this
     * is an integer division with rounding towards 0. To do this portably
     * in C, we shift after obtaining the absolute value.
     */
    if (temp < 0)
        temp = -temp;          /* temp is abs value of input */
    temp >>= A1;                /* apply the point transform */
    absvalues[k] = temp;        /* save abs value for main pass */
    if (temp == 1)
        EOB = k;               /* EOB = index of last newly-nonzero coef */
}

/* Encode the AC coefficients per section G.1.2.3, fig. G.7 */

r = 0;                        /* r = run length of zeros */
BR = 0;                        /* BR = count of buffered bits added now */
BR_buffer = entropy->bit_buffer + entropy->BE; /* Append bits to buffer */

for (k = cinfo->Ss; k <= Se; k++) {
    if ((temp = absvalues[k]) == 0) {
        r++;
        continue;
    }

    /* Emit any required ZRLs, but not if they can be folded into EOB */
    while (r > 15 && k <= EOB) {
        /* emit any pending EOBRUN and the BE correction bits */
        emit_eobrun(entropy);
        /* Emit ZRL */
        emit_symbol(entropy, entropy->ac_tbl_no, 0xF0);
        r -= 16;
        /* Emit buffered correction bits that must be associated with ZRL */
        emit_buffered_bits(entropy, BR_buffer, BR);
        BR_buffer = entropy->bit_buffer; /* BE bits are gone now */
        BR = 0;
    }

    /* If the coef was previously nonzero, it only needs a correction bit.
     * NOTE: a straight translation of the spec's figure G.7 would suggest
     * that we also need to test r > 15. But if r > 15, we can only get here
     * if k > EOB, which implies that this coefficient is not 1.
     */
    if (temp > 1) {
        /* The correction bit is the next bit of the absolute value. */
        BR_buffer[BR++] = (char) (temp & 1);
        continue;
    }

    /* Emit any pending EOBRUN and the BE correction bits */
    emit_eobrun(entropy);

    /* Count/emit Huffman symbol for run length / number of bits */
    emit_symbol(entropy, entropy->ac_tbl_no, (r << 4) + 1);

    /* Emit output bit for newly-nonzero coef */
    temp = ((*block)[jpeg_natural_order[k]] < 0) ? 0 : 1;
    emit_bits(entropy, (unsigned int) temp, 1);

    /* Emit buffered correction bits that must be associated with this code */
    emit_buffered_bits(entropy, BR_buffer, BR);
    BR_buffer = entropy->bit_buffer; /* BE bits are gone now */
    BR = 0;
    r = 0;                /* reset zero run length */
}

if (r > 0 || BR > 0) { /* If there are trailing zeroes, */
    entropy->EOBRUN++;    /* count an EOB */
    entropy->BE += BR;    /* concat my correction bits to older ones */
    /* We force out the EOB if we risk either:
     * 1. overflow of the EOB counter;
     * 2. overflow of the correction bit buffer during the next MCU.
     */
    if (entropy->EOBRUN == 0x7FFF || entropy->BE > (MAX_CORR_BITS-DCTSIZE2+1))

```

```

    emit_eobrun(entropy);
}

cinfo->dest->next_output_byte = entropy->next_output_byte;
cinfo->dest->free_in_buffer = entropy->free_in_buffer;

/* Update restart-interval state too */
if (cinfo->restart_interval) {
    if (entropy->restarts_to_go == 0) {
        entropy->restarts_to_go = cinfo->restart_interval;
        entropy->next_restart_num++;
        entropy->next_restart_num &= 7;
    }
    entropy->restarts_to_go--;
}

return TRUE;
}

/*
 * Finish up at the end of a Huffman-compressed progressive scan.
 */

METHODDEF(void)
finish_pass_phuff (j_compress_ptr cinfo)
{
    phuff_entropy_ptr entropy = (phuff_entropy_ptr) cinfo->entropy;

    entropy->next_output_byte = cinfo->dest->next_output_byte;
    entropy->free_in_buffer = cinfo->dest->free_in_buffer;

    /* Flush out any buffered data */
    emit_eobrun(entropy);
    flush_bits(entropy);

    cinfo->dest->next_output_byte = entropy->next_output_byte;
    cinfo->dest->free_in_buffer = entropy->free_in_buffer;
}

/*
 * Finish up a statistics-gathering pass and create the new Huffman tables.
 */

METHODDEF(void)
finish_pass_gather_phuff (j_compress_ptr cinfo)
{
    phuff_entropy_ptr entropy = (phuff_entropy_ptr) cinfo->entropy;
    boolean is_DC_band;
    int ci, tbl;
    jpeg_component_info * compptr;
    HUFF_TBL **htblptr;
    boolean did[NUM_HUFF_TBLS];

    /* Flush out buffered data (all we care about is counting the EOB symbol) */
    emit_eobrun(entropy);

    is_DC_band = (cinfo->Ss == 0);

    /* It's important not to apply jpeg_gen_optimal_table more than once
     * per table, because it clobbers the input frequency counts!
     */
    MEMZERO(did, sizeof(did));

    for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
        compptr = cinfo->cur_comp_info[ci];
        if (is_DC_band) {
            if (cinfo->Ah != 0) /* DC refinement needs no table */
                continue;
            tbl = compptr->dc_tbl_no;
        } else {
            tbl = compptr->ac_tbl_no;
        }
        if (! did[tbl]) {
            if (is_DC_band)
                htblptr = & cinfo->dc_huff_tbl_ptrs[tbl];
            else
                htblptr = & cinfo->ac_huff_tbl_ptrs[tbl];
            if (*htblptr == NULL)

```

```

        *htblptr = jpeg_alloc_huff_table((j_common_ptr) cinfo);
        jpeg_gen_optimal_table(cinfo, *htblptr, entropy->count_ptrs[tbl]);
        did[tbl] = TRUE;
    }
}

/*
 * Module initialization routine for progressive Huffman entropy encoding.
 */

GLOBAL(void)
jinit_phuff_encoder (j_compress_ptr cinfo)
{
    phuff_entropy_ptr entropy;
    int i;

    entropy = (phuff_entropy_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            SIZEOF(phuff_entropy_encoder));
    cinfo->entropy = (struct jpeg_entropy_encoder *) entropy;
    entropy->pub.start_pass = start_pass_phuff;

    /* Mark tables unallocated */
    for (i = 0; i < NUM_HUFF_TBLS; i++) {
        entropy->derived_tbls[i] = NULL;
        entropy->count_ptrs[i] = NULL;
    }
    entropy->bit_buffer = NULL; /* needed only in AC refinement scan */
}

#endif /* C_PROGRESSIVE_SUPPORTED */

```

```

/*
 * jcprepct.c
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains the compression preprocessing controller.
 * This controller manages the color conversion, downsampling,
 * and edge expansion steps.
 *
 * Most of the complexity here is associated with buffering input rows
 * as required by the downsampler. See the comments at the head of
 * jcsample.c for the downsampler's needs.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/* At present, jcsample.c can request context rows only for smoothing.
 * In the future, we might also need context rows for CCIR601 sampling
 * or other more-complex downsampling procedures. The code to support
 * context rows should be compiled only if needed.
 */
#ifdef INPUT_SMOOTHING_SUPPORTED
#define CONTEXT_ROWS_SUPPORTED
#endif

/* For the simple (no-context-row) case, we just need to buffer one
 * row group's worth of pixels for the downsampling step. At the bottom of
 * the image, we pad to a full row group by replicating the last pixel row.
 * The downsampler's last output row is then replicated if needed to pad
 * out to a full iMCU row.
 *
 * When providing context rows, we must buffer three row groups' worth of
 * pixels. Three row groups are physically allocated, but the row pointer
 * arrays are made five row groups high, with the extra pointers above and
 * below "wrapping around" to point to the last and first real row groups.
 * This allows the downsampler to access the proper context rows.
 * At the top and bottom of the image, we create dummy context rows by
 * copying the first or last real pixel row. This copying could be avoided
 * by pointer hacking as is done in jdmainct.c, but it doesn't seem worth the
 * trouble on the compression side.
 */

/* Private buffer controller object */
typedef struct {
  struct jpeg_c_prep_controller pub; /* public fields */

  /* Downsampling input buffer. This buffer holds color-converted data
   * until we have enough to do a downsample step.
   */
  JSAMPARRAY color_buf[MAX_COMPONENTS];

  JDIMENSION rows_to_go; /* counts rows remaining in source image */
  int next_buf_row; /* index of next row to store in color_buf */

#ifdef CONTEXT_ROWS_SUPPORTED /* only needed for context case */
  int this_row_group; /* starting row index of group to process */
  int next_buf_stop; /* downsample when we reach this index */
#endif
} my_prep_controller;

typedef my_prep_controller * my_prep_ptr;

/*
 * Initialize for a processing pass.
 */
METHODDEF(void)
start_pass_prep (j_compress_ptr cinfo, J_BUF_MODE pass_mode)
{
  my_prep_ptr prep = (my_prep_ptr) cinfo->prep;

```

```

if (pass_mode != JBUF_PASS_THRU)
    ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);

/* Initialize total-height counter for detecting bottom of image */
prep->rows_to_go = cinfo->image_height;
/* Mark the conversion buffer empty */
prep->next_buf_row = 0;
#ifdef CONTEXT_ROWS_SUPPORTED
/* Preset additional state variables for context mode.
 * These aren't used in non-context mode, so we needn't test which mode.
 */
prep->this_row_group = 0;
/* Set next_buf_stop to stop after two row groups have been read in. */
prep->next_buf_stop = 2 * cinfo->max_v_samp_factor;
#endif
}

/*
 * Expand an image vertically from height input_rows to height output_rows,
 * by duplicating the bottom row.
 */

LOCAL(void)
expand_bottom_edge (JSAMPARRAY image_data, JDIMENSION num_cols,
                    int input_rows, int output_rows)
{
    register int row;

    for (row = input_rows; row < output_rows; row++) {
        jcopy_sample_rows(image_data, input_rows-1, image_data, row,
                          1, num_cols);
    }
}

/*
 * Process some data in the simple no-context case.
 *
 * Preprocessor output data is counted in "row groups". A row group
 * is defined to be v_samp_factor sample rows of each component.
 * Downsampling will produce this much data from each max_v_samp_factor
 * input rows.
 */

METHODDEF(void)
pre_process_data (j_compress_ptr cinfo,
                  JSAMPARRAY input_buf, JDIMENSION *in_row_ctr,
                  JDIMENSION in_rows_avail,
                  JSAMPIMAGE output_buf, JDIMENSION *out_row_group_ctr,
                  JDIMENSION out_row_groups_avail)
{
    my_prep_ptr prep = (my_prep_ptr) cinfo->prep;
    int numrows, ci;
    JDIMENSION inrows;
    jpeg_component_info * comp_ptr;

    while (*in_row_ctr < in_rows_avail &&
           *out_row_group_ctr < out_row_groups_avail) {
        /* Do color conversion to fill the conversion buffer. */
        inrows = in_rows_avail - *in_row_ctr;
        numrows = cinfo->max_v_samp_factor - prep->next_buf_row;
        numrows = (int) MIN((JDIMENSION) numrows, inrows);
        (*cinfo->cconvert->color_convert) (cinfo, input_buf + *in_row_ctr,
                                          prep->color_buf,
                                          (JDIMENSION) prep->next_buf_row,
                                          numrows);

        *in_row_ctr += numrows;
        prep->next_buf_row += numrows;
        prep->rows_to_go -= numrows;
        /* If at bottom of image, pad to fill the conversion buffer. */
        if (prep->rows_to_go == 0 &&
            prep->next_buf_row < cinfo->max_v_samp_factor) {
            for (ci = 0; ci < cinfo->num_components; ci++) {
                expand_bottom_edge(prep->color_buf[ci], cinfo->image_width,
                                  prep->next_buf_row, cinfo->max_v_samp_factor);
            }
            prep->next_buf_row = cinfo->max_v_samp_factor;
        }
    }
}

```



```

/* If we've filled the conversion buffer, empty it. */
if (prep->next_buf_row == cinfo->max_v_samp_factor) {
    (*cinfo->downsample->downsample) (cinfo,
        prep->color_buf, (JDIMENSION) 0,
        output_buf, *out_row_group_ctr);
    prep->next_buf_row = 0;
    (*out_row_group_ctr)++;
}
/* If at bottom of image, pad the output to a full iMCU height.
 * Note we assume the caller is providing a one-iMCU-height output buffer!
 */
if (prep->rows_to_go == 0 &&
    *out_row_group_ctr < out_row_groups_avail) {
    for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
        ci++, compptr++) {
        expand_bottom_edge(output_buf[ci],
            compptr->width_in_blocks * DCTSIZE,
            (int) (*out_row_group_ctr * compptr->v_samp_factor),
            (int) (out_row_groups_avail * compptr->v_samp_factor));
    }
    *out_row_group_ctr = out_row_groups_avail;
    break; /* can exit outer loop without test */
}
}
}

```

```

#ifdef CONTEXT_ROWS_SUPPORTED

```

```

/*
 * Process some data in the context case.
 */
METHODDEF(void)
pre_process_context (j_compress_ptr cinfo,
    JSAMPARRAY input_buf, JDIMENSION *in_row_ctr,
    JDIMENSION in_rows_avail,
    JSAMPIMAGE output_buf, JDIMENSION *out_row_group_ctr,
    JDIMENSION out_row_groups_avail)
{
    my_prep_ptr prep = (my_prep_ptr) cinfo->prep;
    int numrows, ci;
    int buf_height = cinfo->max_v_samp_factor * 3;
    JDIMENSION inrows;

    while (*out_row_group_ctr < out_row_groups_avail) {
        if (*in_row_ctr < in_rows_avail) {
            /* Do color conversion to fill the conversion buffer. */
            inrows = in_rows_avail - *in_row_ctr;
            numrows = prep->next_buf_stop - prep->next_buf_row;
            numrows = (int) MIN((JDIMENSION) numrows, inrows);
            (*cinfo->cconvert->color_convert) (cinfo, input_buf + *in_row_ctr,
                prep->color_buf,
                (JDIMENSION) prep->next_buf_row,
                numrows);
            /* Pad at top of image, if first time through */
            if (prep->rows_to_go == cinfo->image_height) {
                for (ci = 0; ci < cinfo->num_components; ci++) {
                    int row;
                    for (row = 1; row <= cinfo->max_v_samp_factor; row++) {
                        jcopy_sample_rows(prep->color_buf[ci], 0,
                            prep->color_buf[ci], -row,
                            1, cinfo->image_width);
                    }
                }
                *in_row_ctr += numrows;
                prep->next_buf_row += numrows;
                prep->rows_to_go -= numrows;
            } else {
                /* Return for more data, unless we are at the bottom of the image. */
                if (prep->rows_to_go != 0)
                    break;
                /* When at bottom of image, pad to fill the conversion buffer. */
                if (prep->next_buf_row < prep->next_buf_stop) {
                    for (ci = 0; ci < cinfo->num_components; ci++) {
                        expand_bottom_edge(prep->color_buf[ci], cinfo->image_width,
                            prep->next_buf_row, prep->next_buf_stop);
                    }
                }
                prep->next_buf_row = prep->next_buf_stop;
            }
        }
        *out_row_group_ctr++;
    }
}

```

```

    }
}
/* If we've gotten enough data, downsample a row group. */
if (prep->next_buf_row == prep->next_buf_stop) {
    (*cinfo->downsample->downsample) (cinfo,
        prep->color_buf,
        (JDIMENSION) prep->this_row_group,
        output_buf, *out_row_group_ctr);
    (*out_row_group_ctr)++;
    /* Advance pointers with wraparound as necessary. */
    prep->this_row_group += cinfo->max_v_samp_factor;
    if (prep->this_row_group >= buf_height)
        prep->this_row_group = 0;
    if (prep->next_buf_row >= buf_height)
        prep->next_buf_row = 0;
    prep->next_buf_stop = prep->next_buf_row + cinfo->max_v_samp_factor;
}
}

/*
 * Create the wrapped-around downsampling input buffer needed for context mode.
 */

LOCAL(void)
create_context_buffer (j_compress_ptr cinfo)
{
    my_prep_ptr prep = (my_prep_ptr) cinfo->prep;
    int rgroup_height = cinfo->max_v_samp_factor;
    int ci, i;
    jpeg_component_info * compptr;
    JSAMPARRAY true_buffer, fake_buffer;

    /* Grab enough space for fake row pointers for all the components;
     * we need five row groups' worth of pointers for each component.
     */
    fake_buffer = (JSAMPARRAY)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            (cinfo->num_components * 5 * rgroup_height) *
            sizeof(JSAMPROW));

    for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
        ci++, compptr++) {
        /* Allocate the actual buffer space (3 row groups) for this component.
         * We make the buffer wide enough to allow the downsampler to edge-expand
         * horizontally within the buffer, if it so chooses.
         */
        true_buffer = (*cinfo->mem->alloc_sarray)
            ((j_common_ptr) cinfo, JPOOL_IMAGE,
            (JDIMENSION) (((long) compptr->width_in_blocks * DCTSIZE *
                cinfo->max_h_samp_factor) / compptr->h_samp_factor),
            (JDIMENSION) (3 * rgroup_height));
        /* Copy true buffer row pointers into the middle of the fake row array */
        MEMCOPY(fake_buffer + rgroup_height, true_buffer,
            3 * rgroup_height * sizeof(JSAMPROW));
        /* Fill in the above and below wraparound pointers */
        for (i = 0; i < rgroup_height; i++) {
            fake_buffer[i] = true_buffer[2 * rgroup_height + i];
            fake_buffer[4 * rgroup_height + i] = true_buffer[i];
        }
        prep->color_buf[ci] = fake_buffer + rgroup_height;
        fake_buffer += 5 * rgroup_height; /* point to space for next component */
    }
}

#endif /* CONTEXT_ROWS_SUPPORTED */

/*
 * Initialize preprocessing controller.
 */

GLOBAL(void)
jinit_c_prep_controller (j_compress_ptr cinfo, boolean need_full_buffer)
{
    my_prep_ptr prep;
    int ci;
    jpeg_component_info * compptr;

```

```

if (need_full_buffer)      /* safety check */
    ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);

prep = (my_prep_ptr)
    (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
        SIZEOF(my_prep_controller));
cinfo->prep = (struct jpeg_c_prep_controller *) prep;
prep->pub.start_pass = start_pass_prep;

/* Allocate the color conversion buffer.
 * We make the buffer wide enough to allow the downsampler to edge-expand
 * horizontally within the buffer, if it so chooses.
 */
if (cinfo->downsample->need_context_rows) {
    /* Set up to provide context rows */
#ifdef CONTEXT_ROWS_SUPPORTED
    prep->pub.pre_process_data = pre_process_context;
    create_context_buffer(cinfo);
#else
    ERREXIT(cinfo, JERR_NOT_COMPILED);
#endif
} else {
    /* No context, just make it tall enough for one row group */
    prep->pub.pre_process_data = pre_process_data;
    for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
        ci++, compptr++) {
        prep->color_buf[ci] = (*cinfo->mem->alloc_sarray)
            ((j_common_ptr) cinfo, JPOOL_IMAGE,
            (JDIMENSION) ((long) compptr->width_in_blocks * DCTSIZE *
                cinfo->max_h_samp_factor) / compptr->h_samp_factor),
            (JDIMENSION) cinfo->max_v_samp_factor);
    }
}

```

```

/*
 * jcsample.c
 *
 * Copyright (C) 1991-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains downsampling routines.
 *
 * Downsampling input data is counted in "row groups".  A row group
 * is defined to be max_v_samp_factor pixel rows of each component,
 * from which the downsampler produces v_samp_factor sample rows.
 * A single row group is processed in each call to the downsampler module.
 *
 * The downsampler is responsible for edge-expansion of its output data
 * to fill an integral number of DCT blocks horizontally.  The source buffer
 * may be modified if it is helpful for this purpose (the source buffer is
 * allocated wide enough to correspond to the desired output width).
 * The caller (the prep controller) is responsible for vertical padding.
 *
 * The downsampler may request "context rows" by setting need_context_rows
 * during startup.  In this case, the input arrays will contain at least
 * one row group's worth of pixels above and below the passed-in data;
 * the caller will create dummy rows at image top and bottom by replicating
 * the first or last real pixel row.
 *
 * An excellent reference for image resampling is
 *   Digital Image Warping, George Wolberg, 1990.
 *   Pub. by IEEE Computer Society Press, Los Alamitos, CA. ISBN 0-8186-8944-7.
 *
 * The downsampling algorithm used here is a simple average of the source
 * pixels covered by the output pixel.  The hi-falutin sampling literature
 * refers to this as a "box filter".  In general the characteristics of a box
 * filter are not very good, but for the specific cases we normally use (1:1
 * and 2:1 ratios) the box is equivalent to a "triangle filter" which is not
 * nearly so bad.  If you intend to use other sampling ratios, you'd be well
 * advised to improve this code.
 *
 * A simple input-smoothing capability is provided.  This is mainly intended
 * for cleaning up color-dithered GIF input files (if you find it inadequate,
 * we suggest using an external filtering program such as pnmconvol).  When
 * enabled, each input pixel P is replaced by a weighted sum of itself and its
 * eight neighbors.  P's weight is 1-8*SF and each neighbor's weight is SF,
 * where SF = (smoothing_factor / 1024).
 * Currently, smoothing is only supported for 2h2v sampling factors.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/* Pointer to routine to downsample a single component */
typedef JMETHOD(void, downsample1_ptr,
  (j_compress_ptr cinfo, jpeg_component_info * comp_ptr,
   JSAMPARRAY input_data, JSAMPARRAY output_data));

/* Private subobject */

typedef struct {
  struct jpeg_downsampler pub; /* public fields */

  /* Downsampling method pointers, one per component */
  downsample1_ptr methods[MAX_COMPONENTS];
} my_downsampler;

typedef my_downsampler * my_downsample_ptr;

/*
 * Initialize for a downsampling pass.
 */

METHODDEF(void)
start_pass_downsample (j_compress_ptr cinfo)
{
  /* no work for now */
}

```

```

/*
 * Expand a component horizontally from width input_cols to width output_cols,
 * by duplicating the rightmost samples.
 */

LOCAL(void)
expand_right_edge (JSAMPARRAY image_data, int num_rows,
                  JDIMENSION input_cols, JDIMENSION output_cols)
{
    register JSAMPROW ptr;
    register JSAMPLE pixval;
    register int count;
    int row;
    int numcols = (int) (output_cols - input_cols);

    if (numcols > 0) {
        for (row = 0; row < num_rows; row++) {
            ptr = image_data[row] + input_cols;
            pixval = ptr[-1]; /* don't need GETJSAMPLE() here */
            for (count = numcols; count > 0; count--)
                *ptr++ = pixval;
        }
    }

/*
 * Do downsampling for a whole row group (all components).
 *
 * In this version we simply downsample each component independently.
 */

METHODDEF(void)
se_dsample (j_compress_ptr cinfo,
            JSAMPIMAGE input_buf, JDIMENSION in_row_index,
            JSAMPIMAGE output_buf, JDIMENSION out_row_group_index)
{
    my_downsample_ptr downsample = (my_downsample_ptr) cinfo->downsample;
    int ci;
    jpeg_component_info * compptr;
    JSAMPARRAY in_ptr, out_ptr;

    for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
         ci++, compptr++) {
        in_ptr = input_buf[ci] + in_row_index;
        out_ptr = output_buf[ci] + (out_row_group_index * compptr->v_samp_factor);
        (*downsample->methods[ci]) (cinfo, compptr, in_ptr, out_ptr);
    }
}

/*
 * Downsample pixel values of a single component.
 * One row group is processed per call.
 * This version handles arbitrary integral sampling ratios, without smoothing.
 * Note that this version is not actually used for customary sampling ratios.
 */

METHODDEF(void)
int_downsample (j_compress_ptr cinfo, jpeg_component_info * compptr,
               JSAMPARRAY input_data, JSAMPARRAY output_data)
{
    int inrow, outrow, h_expand, v_expand, numpix, numpix2, h, v;
    JDIMENSION outcol, outcol_h; /* outcol_h == outcol*h_expand */
    JDIMENSION output_cols = compptr->width_in_blocks * DCTSIZE;
    JSAMPROW inptr, outptr;
    INT32 outvalue;

    h_expand = cinfo->max_h_samp_factor / compptr->h_samp_factor;
    v_expand = cinfo->max_v_samp_factor / compptr->v_samp_factor;
    numpix = h_expand * v_expand;
    numpix2 = numpix/2;

    /* Expand input data enough to let all the output samples be generated
     * by the standard loop. Special-casing padded output would be more
     * efficient.
     */
    expand_right_edge(input_data, cinfo->max_v_samp_factor,
                    cinfo->image_width, output_cols * h_expand);
}

```

```

inrow = 0;
for (outrow = 0; outrow < compptr->v_samp_factor; outrow++) {
    outptr = output_data[outrow];
    for (outcol = 0, outcol_h = 0; outcol < output_cols;
        outcol++, outcol_h += h_expand) {
        outvalue = 0;
        for (v = 0; v < v_expand; v++) {
            inptr = input_data[inrow+v] + outcol_h;
            for (h = 0; h < h_expand; h++) {
                outvalue += (INT32) GETJSAMPLE(*inptr++);
            }
        }
        *outptr++ = (JSAMPLE) ((outvalue + numpix2) / numpix);
    }
    inrow += v_expand;
}

/*
 * Downsample pixel values of a single component.
 * This version handles the special case of a full-size component,
 * without smoothing.
 */

METHODDEF(void)
fullsize_downsample (j_compress_ptr cinfo, jpeg_component_info * compptr,
    JSAMPARRAY input_data, JSAMPARRAY output_data)
{
    /* Copy the data */
    jcopy_sample_rows(input_data, 0, output_data, 0,
        cinfo->max_v_samp_factor, cinfo->image_width);
    /* Edge-expand */
    expand_right_edge(output_data, cinfo->max_v_samp_factor,
        cinfo->image_width, compptr->width_in_blocks * DCTSIZE);
}

/*
 * Downsample pixel values of a single component.
 * This version handles the common case of 2:1 horizontal and 1:1 vertical,
 * without smoothing.
 *
 * A note about the "bias" calculations: when rounding fractional values to
 * integer, we do not want to always round 0.5 up to the next integer.
 * If we did that, we'd introduce a noticeable bias towards larger values.
 * Instead, this code is arranged so that 0.5 will be rounded up or down at
 * alternate pixel locations (a simple ordered dither pattern).
 */

METHODDEF(void)
h2v1_downsample (j_compress_ptr cinfo, jpeg_component_info * compptr,
    JSAMPARRAY input_data, JSAMPARRAY output_data)
{
    int outrow;
    JDIMENSION outcol;
    JDIMENSION output_cols = compptr->width_in_blocks * DCTSIZE;
    register JSAMPROW inptr, outptr;
    register int bias;

    /* Expand input data enough to let all the output samples be generated
     * by the standard loop. Special-casing padded output would be more
     * efficient.
     */
    expand_right_edge(input_data, cinfo->max_v_samp_factor,
        cinfo->image_width, output_cols * 2);

    for (outrow = 0; outrow < compptr->v_samp_factor; outrow++) {
        outptr = output_data[outrow];
        inptr = input_data[outrow];
        bias = 0; /* bias = 0,1,0,1,... for successive samples */
        for (outcol = 0; outcol < output_cols; outcol++) {
            *outptr++ = (JSAMPLE) ((GETJSAMPLE(*inptr) + GETJSAMPLE(inptr[1])
                + bias) >> 1);
            bias ^= 1; /* 0=>1, 1=>0 */
            inptr += 2;
        }
    }
}

```

```

/*
 * Downsample pixel values of a single component.
 * This version handles the standard case of 2:1 horizontal and 2:1 vertical,
 * without smoothing.
 */

```

```

METHODDEF(void)
h2v2_downsample (j_compress_ptr cinfo, jpeg_component_info * compptr,
                 JSAMPARRAY input_data, JSAMPARRAY output_data)
{
    int inrow, outrow;
    JDIMENSION outcol;
    JDIMENSION output_cols = compptr->width_in_blocks * DCTSIZE;
    register JSAMPROW inptr0, inptr1, outptr;
    register int bias;

    /* Expand input data enough to let all the output samples be generated
     * by the standard loop. Special-casing padded output would be more
     * efficient.
     */
    expand_right_edge(input_data, cinfo->max_v_samp_factor,
                     cinfo->image_width, output_cols * 2);

    inrow = 0;
    for (outrow = 0; outrow < compptr->v_samp_factor; outrow++) {
        outptr = output_data[outrow];
        inptr0 = input_data[inrow];
        inptr1 = input_data[inrow+1];
        bias = 1; /* bias = 1,2,1,2,... for successive samples */
        for (outcol = 0; outcol < output_cols; outcol++) {
            *outptr++ = (JSAMPLE) ((GETJSAMPLE(*inptr0) + GETJSAMPLE(inptr0[1]) +
                                   GETJSAMPLE(*inptr1) + GETJSAMPLE(inptr1[1])
                                   + bias) >> 2);
            bias ^= 3; /* 1=>2, 2=>1 */
            inptr0 += 2; inptr1 += 2;
        }
        inrow += 2;
    }
}

```

```

#ifdef INPUT_SMOOTHING_SUPPORTED

```

```

/*
 * Downsample pixel values of a single component.
 * This version handles the standard case of 2:1 horizontal and 2:1 vertical,
 * with smoothing. One row of context is required.
 */

```

```

METHODDEF(void)
h2v2_smooth_downsample (j_compress_ptr cinfo, jpeg_component_info * compptr,
                       JSAMPARRAY input_data, JSAMPARRAY output_data)
{
    int inrow, outrow;
    JDIMENSION colctr;
    JDIMENSION output_cols = compptr->width_in_blocks * DCTSIZE;
    register JSAMPROW inptr0, inptr1, above_ptr, below_ptr, outptr;
    INT32 membersum, neighsum, memberscale, neighscale;

    /* Expand input data enough to let all the output samples be generated
     * by the standard loop. Special-casing padded output would be more
     * efficient.
     */
    expand_right_edge(input_data - 1, cinfo->max_v_samp_factor + 2,
                     cinfo->image_width, output_cols * 2);

    /* We don't bother to form the individual "smoothed" input pixel values;
     * we can directly compute the output which is the average of the four
     * smoothed values. Each of the four member pixels contributes a fraction
     * (1-8*SF) to its own smoothed image and a fraction SF to each of the three
     * other smoothed pixels, therefore a total fraction (1-5*SF)/4 to the final
     * output. The four corner-adjacent neighbor pixels contribute a fraction
     * SF to just one smoothed pixel, or SF/4 to the final output; while the
     * eight edge-adjacent neighbors contribute SF to each of two smoothed
     * pixels, or SF/2 overall. In order to use integer arithmetic, these
     * factors are scaled by 2^16 = 65536.
     * Also recall that SF = smoothing_factor / 1024.
     */
}

```

```

memberscale = 16384 - cinfo->smoothing_factor * 80; /* scaled (1-5*SF)/4 */
neighscale = cinfo->smoothing_factor * 16; /* scaled SF/4 */

inrow = 0;
for (outrow = 0; outrow < compptr->v_samp_factor; outrow++) {
    outptr = output_data[outrow];
    inptr0 = input_data[inrow];
    inptr1 = input_data[inrow+1];
    above_ptr = input_data[inrow-1];
    below_ptr = input_data[inrow+2];

    /* Special case for first column: pretend column -1 is same as column 0 */
    membersum = GETJSAMPLE(*inptr0) + GETJSAMPLE(inptr0[1]) +
        GETJSAMPLE(*inptr1) + GETJSAMPLE(inptr1[1]);
    neighsum = GETJSAMPLE(*above_ptr) + GETJSAMPLE(above_ptr[1]) +
        GETJSAMPLE(*below_ptr) + GETJSAMPLE(below_ptr[1]) +
        GETJSAMPLE(*inptr0) + GETJSAMPLE(inptr0[2]) +
        GETJSAMPLE(*inptr1) + GETJSAMPLE(inptr1[2]);
    neighsum += neighsum;
    neighsum += GETJSAMPLE(*above_ptr) + GETJSAMPLE(above_ptr[2]) +
        GETJSAMPLE(*below_ptr) + GETJSAMPLE(below_ptr[2]);
    membersum = membersum * memberscale + neighsum * neighscale;
    *outptr++ = (JSAMPLE) ((membersum + 32768) >> 16);
    inptr0 += 2; inptr1 += 2; above_ptr += 2; below_ptr += 2;

    for (colctr = output_cols - 2; colctr > 0; colctr--) {
        /* sum of pixels directly mapped to this output element */
        membersum = GETJSAMPLE(*inptr0) + GETJSAMPLE(inptr0[1]) +
            GETJSAMPLE(*inptr1) + GETJSAMPLE(inptr1[1]);
        /* sum of edge-neighbor pixels */
        neighsum = GETJSAMPLE(*above_ptr) + GETJSAMPLE(above_ptr[1]) +
            GETJSAMPLE(*below_ptr) + GETJSAMPLE(below_ptr[1]) +
            GETJSAMPLE(inptr0[-1]) + GETJSAMPLE(inptr0[2]) +
            GETJSAMPLE(inptr1[-1]) + GETJSAMPLE(inptr1[2]);
        /* The edge-neighbors count twice as much as corner-neighbors */
        neighsum += neighsum;
        /* Add in the corner-neighbors */
        neighsum += GETJSAMPLE(above_ptr[-1]) + GETJSAMPLE(above_ptr[2]) +
            GETJSAMPLE(below_ptr[-1]) + GETJSAMPLE(below_ptr[2]);
        /* form final output scaled up by 2^16 */
        membersum = membersum * memberscale + neighsum * neighscale;
        /* round, descale and output it */
        *outptr++ = (JSAMPLE) ((membersum + 32768) >> 16);
        inptr0 += 2; inptr1 += 2; above_ptr += 2; below_ptr += 2;
    }

    /* Special case for last column */
    membersum = GETJSAMPLE(*inptr0) + GETJSAMPLE(inptr0[1]) +
        GETJSAMPLE(*inptr1) + GETJSAMPLE(inptr1[1]);
    neighsum = GETJSAMPLE(*above_ptr) + GETJSAMPLE(above_ptr[1]) +
        GETJSAMPLE(*below_ptr) + GETJSAMPLE(below_ptr[1]) +
        GETJSAMPLE(inptr0[-1]) + GETJSAMPLE(inptr0[1]) +
        GETJSAMPLE(inptr1[-1]) + GETJSAMPLE(inptr1[1]);
    neighsum += neighsum;
    neighsum += GETJSAMPLE(above_ptr[-1]) + GETJSAMPLE(above_ptr[1]) +
        GETJSAMPLE(below_ptr[-1]) + GETJSAMPLE(below_ptr[1]);
    membersum = membersum * memberscale + neighsum * neighscale;
    *outptr = (JSAMPLE) ((membersum + 32768) >> 16);

    inrow += 2;
}

/*
 * Downsample pixel values of a single component.
 * This version handles the special case of a full-size component,
 * with smoothing. One row of context is required.
 */

METHODDEF(void)
fullsize_smooth_downsample (j_compress_ptr cinfo, jpeg_component_info *compptr,
    JSAMPARRAY input_data, JSAMPARRAY output_data)
{
    int outrow;
    JDIMENSION colctr;
    JDIMENSION output_cols = comptr->width_in_blocks * DCTSIZE;
    register JSAMPROW inptr, above_ptr, below_ptr, outptr;
    INT32 membersum, neighsum, memberscale, neighscale;
    int colsum, lastcolsum, nextcolsum;

```



```

/* Expand input data enough to let all the output samples be generated
 * by the standard loop. Special-casing padded output would be more
 * efficient.
 */
expand_right_edge(input_data - 1, cinfo->max_v_samp_factor + 2,
                  cinfo->image_width, output_cols);

/* Each of the eight neighbor pixels contributes a fraction SF to the
 * smoothed pixel, while the main pixel contributes (1-8*SF). In order
 * to use integer arithmetic, these factors are multiplied by 2^16 = 65536.
 * Also recall that SF = smoothing_factor / 1024.
 */

memberscale = 65536L - cinfo->smoothing_factor * 512L; /* scaled 1-8*SF */
neighscale = cinfo->smoothing_factor * 64; /* scaled SF */

for (outrow = 0; outrow < compptr->v_samp_factor; outrow++) {
    outptr = output_data[outrow];
    inptr = input_data[outrow];
    above_ptr = input_data[outrow-1];
    below_ptr = input_data[outrow+1];

    /* Special case for first column */
    colsum = GETJSAMPLE(*above_ptr++) + GETJSAMPLE(*below_ptr++) +
              GETJSAMPLE(*inptr);
    membersum = GETJSAMPLE(*inptr++);
    nextcolsum = GETJSAMPLE(*above_ptr) + GETJSAMPLE(*below_ptr) +
                 GETJSAMPLE(*inptr);
    neighsum = colsum + (colsum - membersum) + nextcolsum;
    membersum = membersum * memberscale + neighsum * neighscale;
    *outptr++ = (JSAMPLE) ((membersum + 32768) >> 16);
    lastcolsum = colsum; colsum = nextcolsum;

    for (colctr = output_cols - 2; colctr > 0; colctr--) {
        membersum = GETJSAMPLE(*inptr++);
        above_ptr++; below_ptr++;
        nextcolsum = GETJSAMPLE(*above_ptr) + GETJSAMPLE(*below_ptr) +
                     GETJSAMPLE(*inptr);
        neighsum = lastcolsum + (colsum - membersum) + nextcolsum;
        membersum = membersum * memberscale + neighsum * neighscale;
        *outptr++ = (JSAMPLE) ((membersum + 32768) >> 16);
        lastcolsum = colsum; colsum = nextcolsum;
    }

    /* Special case for last column */
    membersum = GETJSAMPLE(*inptr);
    neighsum = lastcolsum + (colsum - membersum) + colsum;
    membersum = membersum * memberscale + neighsum * neighscale;
    *outptr = (JSAMPLE) ((membersum + 32768) >> 16);
}

#endif /* INPUT_SMOOTHING_SUPPORTED */

/*
 * Module initialization routine for downsampling.
 * Note that we must select a routine for each component.
 */

GLOBAL(void)
jinit_downsampler (j_compress_ptr cinfo)
{
    my_downsample_ptr downsample;
    int ci;
    jpeg_component_info * compptr;
    boolean smoothok = TRUE;

    downsample = (my_downsample_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                                   SIZEOF(my_downsampler));
    cinfo->downsample = (struct jpeg_downsampler *) downsample;
    downsample->pub.start_pass = start_pass_downsample;
    downsample->pub.downsample = sep_downsample;
    downsample->pub.need_context_rows = FALSE;

    if (cinfo->CCIR601_sampling)
        ERREXIT(cinfo, JERR_CCIR601_NOTIMPL);
}

```

```

/* Verify we can handle the sampling factors, and set up method pointers */
for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
    ci++, compptr++) {
    if (compptr->h_samp_factor == cinfo->max_h_samp_factor &&
        compptr->v_samp_factor == cinfo->max_v_samp_factor) {
#ifdef INPUT_SMOOTHING_SUPPORTED
        if (cinfo->smoothing_factor) {
            downsample->methods[ci] = fullsize_smooth_downsample;
            downsample->pub.need_context_rows = TRUE;
        } else
#endif
        downsample->methods[ci] = fullsize_downsample;
    } else if (compptr->h_samp_factor * 2 == cinfo->max_h_samp_factor &&
        compptr->v_samp_factor == cinfo->max_v_samp_factor) {
        smoothok = FALSE;
        downsample->methods[ci] = h2v1_downsample;
    } else if (compptr->h_samp_factor * 2 == cinfo->max_h_samp_factor &&
        compptr->v_samp_factor * 2 == cinfo->max_v_samp_factor) {
#ifdef INPUT_SMOOTHING_SUPPORTED
        if (cinfo->smoothing_factor) {
            downsample->methods[ci] = h2v2_smooth_downsample;
            downsample->pub.need_context_rows = TRUE;
        } else
#endif
        downsample->methods[ci] = h2v2_downsample;
    } else if ((cinfo->max_h_samp_factor % compptr->h_samp_factor) == 0 &&
        (cinfo->max_v_samp_factor % compptr->v_samp_factor) == 0) {
        smoothok = FALSE;
        downsample->methods[ci] = int_downsample;
    } else
        ERREXIT(cinfo, JERR_FRACT_SAMPLE_NOTIMPL);
}

#ifdef INPUT_SMOOTHING_SUPPORTED
if (cinfo->smoothing_factor && !smoothok)
    TRACEMS(cinfo, 0, JTRC_SMOOTH_NOTIMPL);
#endif
}

```

```

/*
 * jctrans.c
 *
 * Copyright (C) 1995-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains library routines for transcoding compression,
 * that is, writing raw DCT coefficient arrays to an output JPEG file.
 * The routines in jcapimin.c will also be needed by a transcoder.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/* Forward declarations */
LOCAL(void) transencode_master_selection
    JPP((j_compress_ptr cinfo, jvirt_barray_ptr * coef_arrays));
LOCAL(void) transencode_coef_controller
    JPP((j_compress_ptr cinfo, jvirt_barray_ptr * coef_arrays));

/*
 * Compression initialization for writing raw-coefficient data.
 * Before calling this, all parameters and a data destination must be set up.
 * Call jpeg_finish_compress() to actually write the data.
 *
 * The number of passed virtual arrays must match cinfo->num_components.
 * Note that the virtual arrays need not be filled or even realized at
 * the time write_coefficients is called; indeed, if the virtual arrays
 * were requested from this compression object's memory manager, they
 * typically will be realized during this routine and filled afterwards.
 */
GLOBAL(void)
jpeg_write_coefficients (j_compress_ptr cinfo, jvirt_barray_ptr * coef_arrays)
{
    if (cinfo->global_state != CSTATE_START)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
    /* Mark all tables to be written */
    jpeg_suppress_tables(cinfo, FALSE);
    /* (Re)initialize error mgr and destination modules */
    (*cinfo->err->reset_error_mgr) ((j_common_ptr) cinfo);
    (*cinfo->dest->init_destination) (cinfo);
    /* Perform master selection of active modules */
    transencode_master_selection(cinfo, coef_arrays);
    /* Wait for jpeg_finish_compress() call */
    cinfo->next_scanline = 0; /* so jpeg_write_marker works */
    cinfo->global_state = CSTATE_WRCOEFS;
}

/*
 * Initialize the compression object with default parameters,
 * then copy from the source object all parameters needed for lossless
 * transcoding. Parameters that can be varied without loss (such as
 * scan script and Huffman optimization) are left in their default states.
 */
GLOBAL(void)
jpeg_copy_critical_parameters (j_decompress_ptr srcinfo,
                              j_compress_ptr dstinfo)
{
    JQUANT_TBL ** qtblptr;
    jpeg_component_info *incomp, *outcomp;
    JQUANT_TBL *c_quant, *slot_quant;
    int tblno, ci, coefi;

    /* Safety check to ensure start_compress not called yet. */
    if (dstinfo->global_state != CSTATE_START)
        ERREXIT1(dstinfo, JERR_BAD_STATE, dstinfo->global_state);
    /* Copy fundamental image dimensions */
    dstinfo->image_width = srcinfo->image_width;
    dstinfo->image_height = srcinfo->image_height;
    dstinfo->input_components = srcinfo->num_components;
    dstinfo->in_color_space = srcinfo->jpeg_color_space;
    /* Initialize all parameters to default values */
    jpeg_set_defaults(dstinfo);
}

```

```

/* jpeg_set_defaults may choose wrong colorspace, eg YCbCr if input is RGB.
 * Fix it to get the right header markers for the image colorspace.
 */
jpeg_set_colorspace(dstinfo, srcinfo->jpeg_color_space);
dstinfo->data_precision = srcinfo->data_precision;
dstinfo->CCIR601_sampling = srcinfo->CCIR601_sampling;
/* Copy the source's quantization tables. */
for (tblno = 0; tblno < NUM_QUANT_TBLS; tblno++) {
    if (srcinfo->quant_tbl_ptrs[tblno] != NULL) {
        qtblptr = &dstinfo->quant_tbl_ptrs[tblno];
        if (*qtblptr == NULL)
            *qtblptr = jpeg_alloc_quant_table((j_common_ptr) dstinfo);
        MEMCOPY((*qtblptr)->quantval,
            srcinfo->quant_tbl_ptrs[tblno]->quantval,
            SIZEOF((*qtblptr)->quantval));
        (*qtblptr)->sent_table = FALSE;
    }
}
/* Copy the source's per-component info.
 * Note we assume jpeg_set_defaults has allocated the dest comp_info array.
 */
dstinfo->num_components = srcinfo->num_components;
if (dstinfo->num_components < 1 || dstinfo->num_components > MAX_COMPONENTS)
    ERREXIT2(dstinfo, JERR_COMPONENT_COUNT, dstinfo->num_components,
        MAX_COMPONENTS);
for (ci = 0, incomp = srcinfo->comp_info, outcomp = dstinfo->comp_info;
    ci < dstinfo->num_components; ci++, incomp++, outcomp++) {
    outcomp->component_id = incomp->component_id;
    outcomp->h_samp_factor = incomp->h_samp_factor;
    outcomp->v_samp_factor = incomp->v_samp_factor;
    outcomp->quant_tbl_no = incomp->quant_tbl_no;
    /* Make sure saved quantization table for component matches the qtable
     * slot. If not, the input file re-used this qtable slot.
     * IJG encoder currently cannot duplicate this.
     */
    tblno = outcomp->quant_tbl_no;
    if (tblno < 0 || tblno >= NUM_QUANT_TBLS ||
        srcinfo->quant_tbl_ptrs[tblno] == NULL)
        ERREXIT1(dstinfo, JERR_NO_QUANT_TABLE, tblno);
    slot_quant = srcinfo->quant_tbl_ptrs[tblno];
    c_quant = incomp->quant_table;
    if (c_quant != NULL) {
        for (coefi = 0; coefi < DCTSIZE2; coefi++) {
            if (c_quant->quantval[coefi] != slot_quant->quantval[coefi])
                ERREXIT1(dstinfo, JERR_MISMATCHED_QUANT_TABLE, tblno);
        }
    }
    /* Note: we do not copy the source's Huffman table assignments;
     * instead we rely on jpeg_set_colorspace to have made a suitable choice.
     */
    /* Also copy JFIF version and resolution information, if available.
     * Strictly speaking this isn't "critical" info, but it's nearly
     * always appropriate to copy it if available. In particular,
     * if the application chooses to copy JFIF 1.02 extension markers from
     * the source file, we need to copy the version to make sure we don't
     * emit a file that has 1.02 extensions but a claimed version of 1.01.
     * We will *not*, however, copy version info from mislabeled "2.01" files.
     */
    if (srcinfo->saw_JFIF_marker) {
        if (srcinfo->JFIF_major_version == 1) {
            dstinfo->JFIF_major_version = srcinfo->JFIF_major_version;
            dstinfo->JFIF_minor_version = srcinfo->JFIF_minor_version;
        }
        dstinfo->density_unit = srcinfo->density_unit;
        dstinfo->X_density = srcinfo->X_density;
        dstinfo->Y_density = srcinfo->Y_density;
    }
}

/*
 * Master selection of compression modules for transcoding.
 * This substitutes for jcinit.c's initialization of the full compressor.
 */

LOCAL(void)
transencode_master_selection (j_compress_ptr cinfo,
    jvirt_barray_ptr * coef_arrays)
{

```

```

/* Although we don't actually use input_components for transcoding,
 * jcmaster.c's initial_setup will complain if input_components is 0.
 */
cinfo->input_components = 1;
/* Initialize master control (includes parameter checking/processing) */
jinit_c_master_control(cinfo, TRUE /* transcode only */);

/* Entropy encoding: either Huffman or arithmetic coding. */
if (cinfo->arith_code) {
    ERREXIT(cinfo, JERR_ARITH_NOTIMPL);
} else {
    if (cinfo->progressive_mode) {
#ifdef C_PROGRESSIVE_SUPPORTED
        jinit_phuff_encoder(cinfo);
    #else
        ERREXIT(cinfo, JERR_NOT_COMPILED);
    #endif
    } else
        jinit_huff_encoder(cinfo);
}

/* We need a special coefficient buffer controller. */
transcode_coef_controller(cinfo, coef_arrays);

jinit_marker_writer(cinfo);

/* We can now tell the memory manager to allocate virtual arrays. */
(*cinfo->mem->realize_virt_arrays) ((j_common_ptr) cinfo);

/* Write the datastream header (SOI, JFIF) immediately.
 * Frame and scan headers are postponed till later.
 * This lets application insert special markers after the SOI.
 */
(*cinfo->marker->write_file_header) (cinfo);

/*
 * The rest of this file is a special implementation of the coefficient
 * buffer controller. This is similar to jcccoefct.c, but it handles only
 * output from presupplied virtual arrays. Furthermore, we generate any
 * dummy padding blocks on-the-fly rather than expecting them to be present
 * in the arrays.
 */

/* Private buffer controller object */
typedef struct {
    struct jpeg_coef_controller pub; /* public fields */

    JDIMENSION iMCU_row_num; /* iMCU row # within image */
    JDIMENSION mcu_ctr; /* counts MCUs processed in current row */
    int MCU_vert_offset; /* counts MCU rows within iMCU row */
    int MCU_rows_per_iMCU_row; /* number of such rows needed */

    /* Virtual block array for each component. */
    jvirt_barray_ptr * whole_image;

    /* Workspace for constructing dummy blocks at right/bottom edges. */
    JBLOCKROW dummy_buffer[C_MAX_BLOCKS_IN_MCU];
} my_coef_controller;

typedef my_coef_controller * my_coef_ptr;

LOCAL(void)
start_iMCU_row (j_compress_ptr cinfo)
/* Reset within-iMCU-row counters for a new row */
{
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;

    /* In an interleaved scan, an MCU row is the same as an iMCU row.
     * In a noninterleaved scan, an iMCU row has v_samp_factor MCU rows.
     * But at the bottom of the image, process only what's left.
     */
    if (cinfo->comps_in_scan > 1) {
        coef->MCU_rows_per_iMCU_row = 1;
    } else {
        if (coef->iMCU_row_num < (cinfo->total_iMCU_rows-1))
            coef->MCU_rows_per_iMCU_row = cinfo->cur_comp_info[0]->v_samp_factor;
    }
}

```

```

    else
        coef->MCU_rows_per_iMCU_row = cinfo->cur_comp_info[0]->last_row_height;
    }

    coef->mcu_ctr = 0;
    coef->MCU_vert_offset = 0;
}

/*
 * Initialize for a processing pass.
 */

METHODDEF(void)
start_pass_coef (j_compress_ptr cinfo, J_BUF_MODE pass_mode)
{
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;

    if (pass_mode != JBUF_CRANK_DEST)
        ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);

    coef->iMCU_row_num = 0;
    start_iMCU_row(cinfo);
}

/*
 * Process some data.
 * We process the equivalent of one fully interleaved MCU row ("iMCU" row)
 * per call, ie, v_samp_factor block rows for each component in the scan.
 * The data is obtained from the virtual arrays and fed to the entropy coder.
 * Returns TRUE if the iMCU row is completed, FALSE if suspended.
 * NB: input_buf is ignored; it is likely to be a NULL pointer.
 */

METHODDEF(boolean)
compress_output (j_compress_ptr cinfo, JSAMPIMAGE input_buf)
{
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;
    JDIMENSION MCU_col_num; /* index of current MCU within row */
    JDIMENSION last_MCU_col = cinfo->MCUs_per_row - 1;
    JDIMENSION last_iMCU_row = cinfo->total_iMCU_rows - 1;
    int blkn, ci, xindex, yindex, yoffset, blockcnt;
    JDIMENSION start_col;
    JBLOCKARRAY buffer[MAX_COMPS_IN_SCAN];
    JBLOCKROW MCU_buffer[C_MAX_BLOCKS_IN_MCU];
    JBLOCKROW buffer_ptr;
    jpeg_component_info *comp_ptr;

    /* Align the virtual buffers for the components used in this scan. */
    for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
        comp_ptr = cinfo->cur_comp_info[ci];
        buffer[ci] = (*cinfo->mem->access_virt_barray)
            ((j_common_ptr) cinfo, coef->whole_image[comp_ptr->component_index],
             coef->iMCU_row_num * comp_ptr->v_samp_factor,
             (JDIMENSION) comp_ptr->v_samp_factor, FALSE);
    }

    /* Loop to process one whole iMCU row */
    for (yoffset = coef->MCU_vert_offset; yoffset < coef->MCU_rows_per_iMCU_row;
         yoffset++) {
        for (MCU_col_num = coef->mcu_ctr; MCU_col_num < cinfo->MCUs_per_row;
             MCU_col_num++) {
            /* Construct list of pointers to DCT blocks belonging to this MCU */
            blkn = 0; /* index of current DCT block within MCU */
            for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
                comp_ptr = cinfo->cur_comp_info[ci];
                start_col = MCU_col_num * comp_ptr->MCU_width;
                blockcnt = (MCU_col_num < last_MCU_col) ? comp_ptr->MCU_width
                    : comp_ptr->last_col_width;
                for (yindex = 0; yindex < comp_ptr->MCU_height; yindex++) {
                    if (coef->iMCU_row_num < last_iMCU_row ||
                        yindex+yoffset < comp_ptr->last_row_height) {
                        /* Fill in pointers to real blocks in this row */
                        buffer_ptr = buffer[ci][yindex+yoffset] + start_col;
                        for (xindex = 0; xindex < blockcnt; xindex++)
                            MCU_buffer[blkn++] = buffer_ptr++;
                    } else {
                        /* At bottom of image, need a whole row of dummy blocks */

```

```

    xindex = 0;
}
/* Fill in any dummy blocks needed in this row.
 * Dummy blocks are filled in the same way as in jcccoefct.c:
 * all zeroes in the AC entries, DC entries equal to previous
 * block's DC value. The init routine has already zeroed the
 * AC entries, so we need only set the DC entries correctly.
 */
for (; xindex < compptr->MCU_width; xindex++) {
    MCU_buffer[blkn] = coef->dummy_buffer[blkn];
    MCU_buffer[blkn][0][0] = MCU_buffer[blkn-1][0][0];
    blkn++;
}
}
/* Try to write the MCU. */
if (!(*cinfo->entropy->encode_mcu) (cinfo, MCU_buffer)) {
/* Suspension forced; update state counters and exit */
coef->MCU_vert_offset = yoffset;
coef->mcu_ctr = MCU_col_num;
return FALSE;
}
}
/* Completed an MCU row, but perhaps not an iMCU row */
coef->mcu_ctr = 0;
}
/* Completed the iMCU row, advance counters for next one */
coef->iMCU_row_num++;
start_iMCU_row(cinfo);
return TRUE;
}
}

/*
 * Initialize coefficient buffer controller.
 *
 * Each passed coefficient array must be the right size for that
 * coefficient: width_in_blocks wide and height_in_blocks high,
 * with unitheight at least v_samp_factor.
 */
LOCAL(void)
transencode_coef_controller (j_compress_ptr cinfo,
                             jvirt_barray_ptr * coef_arrays)
{
    my_coef_ptr coef;
    JBLOCKROW buffer;
    int i;

    coef = (my_coef_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                                   sizeof(my_coef_controller));
    cinfo->coef = (struct jpeg_c_coef_controller *) coef;
    coef->pub.start_pass = start_pass_coef;
    coef->pub.compress_data = compress_output;

    /* Save pointer to virtual arrays */
    coef->whole_image = coef_arrays;

    /* Allocate and pre-zero space for dummy DCT blocks. */
    buffer = (JBLOCKROW)
        (*cinfo->mem->alloc_large) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                                   C_MAX_BLOCKS_IN_MCU * sizeof(JBLOCK));
    jzero_far((void *) buffer, C_MAX_BLOCKS_IN_MCU * sizeof(JBLOCK));
    for (i = 0; i < C_MAX_BLOCKS_IN_MCU; i++) {
        coef->dummy_buffer[i] = buffer + i;
    }
}

```

```

/*
 * jdapimin.c
 *
 * Copyright (C) 1994-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains application interface code for the decompression half
 * of the JPEG library.  These are the "minimum" API routines that may be
 * needed in either the normal full-decompression case or the
 * transcoding-only case.
 *
 * Most of the routines intended to be called directly by an application
 * are in this file or in jdapistd.c.  But also see jcomapi.c for routines
 * shared by compression and decompression, and jdtrans.c for the transcoding
 * case.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/*
 * Initialization of a JPEG decompression object.
 * The error manager must already be set up (in case memory manager fails).
 */

GLOBAL(void)
jpeg_CreateDecompress (j_decompress_ptr cinfo, int version, size_t structsize)
{
  int i;

  /* Guard against version mismatches between library and caller. */
  cinfo->mem = NULL; /* so jpeg_destroy knows mem mgr not called */
  if (version != JPEG_LIB_VERSION)
    ERREXIT2(cinfo, JERR_BAD_LIB_VERSION, JPEG_LIB_VERSION, version);
  if (structsize != SIZEOF(struct jpeg_decompress_struct))
    ERREXIT2(cinfo, JERR_BAD_STRUCT_SIZE,
      (int) SIZEOF(struct jpeg_decompress_struct), (int) structsize);

  /* For debugging purposes, we zero the whole master structure.
   * But the application has already set the err pointer, and may have set
   * client_data, so we have to save and restore those fields.
   * Note: if application hasn't set client_data, tools like Purify may
   * complain here.
   */
  struct jpeg_error_mgr * err = cinfo->err;
  void * client_data = cinfo->client_data; /* ignore Purify complaint here */
  MEMZERO(cinfo, SIZEOF(struct jpeg_decompress_struct));
  cinfo->err = err;
  cinfo->client_data = client_data;
}

cinfo->is_decompressor = TRUE;

/* Initialize a memory manager instance for this object */
jinit_memory_mgr((j_common_ptr) cinfo);

/* Zero out pointers to permanent structures. */
cinfo->progress = NULL;
cinfo->src = NULL;

for (i = 0; i < NUM_QUANT_TBLS; i++)
  cinfo->quant_tbl_ptrs[i] = NULL;

for (i = 0; i < NUM_HUFF_TBLS; i++) {
  cinfo->dc_huff_tbl_ptrs[i] = NULL;
  cinfo->ac_huff_tbl_ptrs[i] = NULL;
}

/* Initialize marker processor so application can override methods
 * for COM, APPn markers before calling jpeg_read_header.
 */
cinfo->marker_list = NULL;
jinit_marker_reader(cinfo);

/* And initialize the overall input controller. */
jinit_input_controller(cinfo);

```



```

/* OK, I'm ready */
cinfo->global_state = DSTATE_START;
}

/*
 * Destruction of a JPEG decompression object
 */

GLOBAL(void)
jpeg_destroy_decompress (j_decompress_ptr cinfo)
{
    jpeg_destroy((j_common_ptr) cinfo); /* use common routine */
}

/*
 * Abort processing of a JPEG decompression operation,
 * but don't destroy the object itself.
 */

GLOBAL(void)
jpeg_abort_decompress (j_decompress_ptr cinfo)
{
    jpeg_abort((j_common_ptr) cinfo); /* use common routine */
}

/*
 * Set default decompression parameters.
 */

LOCAL(void)
default_decompress_parms (j_decompress_ptr cinfo)
{
    /* Guess the input colorspace, and set output colorspace accordingly. */
    /* (Wish JPEG committee had provided a real way to specify this...) */
    /* Note application may override our guesses. */
    switch (cinfo->num_components) {
        case 1:
            cinfo->jpeg_color_space = JCS_GRAYSCALE;
            cinfo->out_color_space = JCS_GRAYSCALE;
            break;

        case 3:
            if (cinfo->saw_JFIF_marker) {
                cinfo->jpeg_color_space = JCS_YCbCr; /* JFIF implies YCbCr */
            } else if (cinfo->saw_Adobe_marker) {
                switch (cinfo->Adobe_transform) {
                    case 0:
                        cinfo->jpeg_color_space = JCS_RGB;
                        break;
                    case 1:
                        cinfo->jpeg_color_space = JCS_YCbCr;
                        break;
                    default:
                        WARNMS1(cinfo, JWRN_ADOBE_XFORM, cinfo->Adobe_transform);
                        cinfo->jpeg_color_space = JCS_YCbCr; /* assume it's YCbCr */
                        break;
                }
            } else {
                /* Saw no special markers, try to guess from the component IDs */
                int cid0 = cinfo->comp_info[0].component_id;
                int cid1 = cinfo->comp_info[1].component_id;
                int cid2 = cinfo->comp_info[2].component_id;

                if (cid0 == 1 && cid1 == 2 && cid2 == 3)
                    cinfo->jpeg_color_space = JCS_YCbCr; /* assume JFIF w/out marker */
                else if (cid0 == 82 && cid1 == 71 && cid2 == 66)
                    cinfo->jpeg_color_space = JCS_RGB; /* ASCII 'R', 'G', 'B' */
                else {
                    TRACE3(cinfo, 1, JTRC_UNKNOWN_IDS, cid0, cid1, cid2);
                    cinfo->jpeg_color_space = JCS_YCbCr; /* assume it's YCbCr */
                }
            }
            /* Always guess RGB is proper output colorspace. */
            cinfo->out_color_space = JCS_RGB;
            break;

        case 4:

```

```

    if (cinfo->saw_Adobe_marker) {
        switch (cinfo->Adobe_transform) {
            case 0:
                cinfo->jpeg_color_space = JCS_CMYK;
                break;
            case 2:
                cinfo->jpeg_color_space = JCS_YCCK;
                break;
            default:
                WARNMS1(cinfo, JWRN_ADOBE_XFORM, cinfo->Adobe_transform);
                cinfo->jpeg_color_space = JCS_YCCK; /* assume it's YCCK */
                break;
        }
    } else {
        /* No special markers, assume straight CMYK. */
        cinfo->jpeg_color_space = JCS_CMYK;
    }
    cinfo->out_color_space = JCS_CMYK;
    break;

default:
    cinfo->jpeg_color_space = JCS_UNKNOWN;
    cinfo->out_color_space = JCS_UNKNOWN;
    break;
}

/* Set defaults for other decompression parameters. */
cinfo->scale_num = 1;      /* 1:1 scaling */
cinfo->scale_denom = 1;
cinfo->output_gamma = 1.0;
cinfo->buffered_image = FALSE;
cinfo->raw_data_out = FALSE;
cinfo->dct_method = JDCT_DEFAULT;
cinfo->do_fancy_upsampling = TRUE;
cinfo->do_block_smoothing = TRUE;
cinfo->quantize_colors = FALSE;
/* We set these in case application only sets quantize_colors. */
cinfo->dither_mode = JDITHER_FS;
#ifdef QUANT_2PASS_SUPPORTED
    cinfo->two_pass_quantize = TRUE;
#else
    cinfo->two_pass_quantize = FALSE;
#endif
cinfo->desired_number_of_colors = 256;
cinfo->colormap = NULL;
/* Initialize for no mode change in buffered-image mode. */
cinfo->enable_1pass_quant = FALSE;
cinfo->enable_external_quant = FALSE;
cinfo->enable_2pass_quant = FALSE;
}

/* Decompression startup: read start of JPEG datastream to see what's there.
 * Need only initialize JPEG object and supply a data source before calling.
 *
 * This routine will read as far as the first SOS marker (ie, actual start of
 * compressed data), and will save all tables and parameters in the JPEG
 * object. It will also initialize the decompression parameters to default
 * values, and finally return JPEG_HEADER_OK. On return, the application may
 * adjust the decompression parameters and then call jpeg_start_decompress.
 * (Or, if the application only wanted to determine the image parameters,
 * the data need not be decompressed. In that case, call jpeg_abort or
 * jpeg_destroy to release any temporary space.)
 * If an abbreviated (tables only) datastream is presented, the routine will
 * return JPEG_HEADER_TABLES_ONLY upon reaching EOI. The application may then
 * re-use the JPEG object to read the abbreviated image datastream(s).
 * It is unnecessary (but OK) to call jpeg_abort in this case.
 * The JPEG_SUSPENDED return code only occurs if the data source module
 * requests suspension of the decompressor. In this case the application
 * should load more source data and then re-call jpeg_read_header to resume
 * processing.
 * If a non-suspending data source is used and require_image is TRUE, then the
 * return code need not be inspected since only JPEG_HEADER_OK is possible.
 *
 * This routine is now just a front end to jpeg_consume_input, with some
 * extra error checking.
 */

```

```
GLOBAL(int)
```

```

jpeg_read_header (j_decompress_ptr cinfo, boolean require_image)
{
    int retcode;

    if (cinfo->global_state != DSTATE_START &&
        cinfo->global_state != DSTATE_INHEADER)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);

    retcode = jpeg_consume_input(cinfo);

    switch (retcode) {
    case JPEG_REACHED_SOS:
        retcode = JPEG_HEADER_OK;
        break;
    case JPEG_REACHED_EOI:
        if (require_image) /* Complain if application wanted an image */
            ERREXIT(cinfo, JERR_NO_IMAGE);
        /* Reset to start state; it would be safer to require the application to
         * call jpeg_abort, but we can't change it now for compatibility reasons.
         * A side effect is to free any temporary memory (there shouldn't be any).
         */
        jpeg_abort((j_common_ptr) cinfo); /* sets state = DSTATE_START */
        retcode = JPEG_HEADER_TABLES_ONLY;
        break;
    case JPEG_SUSPENDED:
        /* no work */
        break;
    }

    return retcode;
}

/*
 * Consume data in advance of what the decompressor requires.
 * This can be called at any time once the decompressor object has
 * been created and a data source has been set up.
 *
 * This routine is essentially a state machine that handles a couple
 * of critical state-transition actions, namely initial setup and
 * transition from header scanning to ready-for-start_decompress.
 * All the actual input is done via the input controller's consume_input
 * method.
 */
GLOBAL(int)
jpeg_consume_input (j_decompress_ptr cinfo)
{
    int retcode = JPEG_SUSPENDED;

    /* NB: every possible DSTATE value should be listed in this switch */
    switch (cinfo->global_state) {
    case DSTATE_START:
        /* Start-of-datastream actions: reset appropriate modules */
        (*cinfo->inputctl->reset_input_controller) (cinfo);
        /* Initialize application's data source module */
        (*cinfo->src->init_source) (cinfo);
        cinfo->global_state = DSTATE_INHEADER;
        /* FALLTHROUGH */
    case DSTATE_INHEADER:
        retcode = (*cinfo->inputctl->consume_input) (cinfo);
        if (retcode == JPEG_REACHED_SOS) { /* Found SOS, prepare to decompress */
            /* Set up default parameters based on header data */
            default_decompress_parms(cinfo);
            /* Set global state: ready for start_decompress */
            cinfo->global_state = DSTATE_READY;
        }
        break;
    case DSTATE_READY:
        /* Can't advance past first SOS until start_decompress is called */
        retcode = JPEG_REACHED_SOS;
        break;
    case DSTATE_PRELOAD:
    case DSTATE_PRESCAN:
    case DSTATE_SCANNING:
    case DSTATE_RAW_OK:
    case DSTATE_BUFIMAGE:
    case DSTATE_BUFPOST:
    case DSTATE_STOPPING:
        retcode = (*cinfo->inputctl->consume_input) (cinfo);
    }
}

```

```

        break;
    default:
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
    }
    return retcode;
}

/*
 * Have we finished reading the input file?
 */

GLOBAL(boolean)
jpeg_input_complete (j_decompress_ptr cinfo)
{
    /* Check for valid jpeg object */
    if (cinfo->global_state < DSTATE_START ||
        cinfo->global_state > DSTATE_STOPPING)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
    return cinfo->inputctl->eoi_reached;
}

/*
 * Is there more than one scan?
 */

GLOBAL(boolean)
jpeg_has_multiple_scans (j_decompress_ptr cinfo)
{
    /* Only valid after jpeg_read_header completes */
    if (cinfo->global_state < DSTATE_READY ||
        cinfo->global_state > DSTATE_STOPPING)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
    return cinfo->inputctl->has_multiple_scans;
}

/*
 * Finish JPEG decompression.
 *
 * This will normally just verify the file trailer and release temp storage.
 * Returns FALSE if suspended. The return value need be inspected only if
 * a suspending data source is used.
 */

GLOBAL(boolean)
jpeg_finish_decompress (j_decompress_ptr cinfo)
{
    if ((cinfo->global_state == DSTATE_SCANNING ||
        cinfo->global_state == DSTATE_RAW_OK) && ! cinfo->buffered_image) {
        /* Terminate final pass of non-buffered mode */
        if (cinfo->output_scanline < cinfo->output_height)
            ERREXIT(cinfo, JERR_TOO_LITTLE_DATA);
        (*cinfo->master->finish_output_pass) (cinfo);
        cinfo->global_state = DSTATE_STOPPING;
    } else if (cinfo->global_state == DSTATE_BUFIMAGE) {
        /* Finishing after a buffered-image operation */
        cinfo->global_state = DSTATE_STOPPING;
    } else if (cinfo->global_state != DSTATE_STOPPING) {
        /* STOPPING = repeat call after a suspension, anything else is error */
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
    }
    /* Read until EOI */
    while (! cinfo->inputctl->eoi_reached) {
        if ((*cinfo->inputctl->consume_input) (cinfo) == JPEG_SUSPENDED)
            return FALSE; /* Suspend, come back later */
    }
    /* Do final cleanup */
    (*cinfo->src->term_source) (cinfo);
    /* We can use jpeg_abort to release memory and reset global_state */
    jpeg_abort((j_common_ptr) cinfo);
    return TRUE;
}

```

```

/*
 * jdapistd.c
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains application interface code for the decompression half
 * of the JPEG library. These are the "standard" API routines that are
 * used in the normal full-decompression case. They are not used by a
 * transcoding-only application. Note that if an application links in
 * jpeg_start_decompress, it will end up linking in the entire decompressor.
 * We thus must separate this file from jdapimin.c to avoid linking the
 * whole decompression library into a transcoder.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/* Forward declarations */
LOCAL(boolean) output_pass_setup JPP((j_decompress_ptr cinfo));

/*
 * Decompression initialization.
 * jpeg_read_header must be completed before calling this.
 *
 * If a multipass operating mode was selected, this will do all but the
 * last pass, and thus may take a great deal of time.
 *
 * Returns FALSE if suspended. The return value need be inspected only if
 * a suspending data source is used.
 */

GLOBAL(boolean)
jpeg_start_decompress (j_decompress_ptr cinfo)
{
  if (cinfo->global_state == DSTATE_READY) {
    /* First call: initialize master control, select active modules */
    jinit_master_decompress(cinfo);
    if (cinfo->buffered_image) {
      /* No more work here; expecting jpeg_start_output next */
      cinfo->global_state = DSTATE_BUFIMAGE;
      return TRUE;
    }
    cinfo->global_state = DSTATE_PRELOAD;
  }
  if (cinfo->global_state == DSTATE_PRELOAD) {
    /* If file has multiple scans, absorb them all into the coef buffer */
    if (cinfo->inputctl->has_multiple_scans) {
#ifdef D_MULTISCAN_FILES_SUPPORTED
      for (;;) {
        int retcode;
        /* Call progress monitor hook if present */
        if (cinfo->progress != NULL)
          (*cinfo->progress->progress_monitor) ((j_common_ptr) cinfo);
        /* Absorb some more input */
        retcode = (*cinfo->inputctl->consume_input) (cinfo);
        if (retcode == JPEG_SUSPENDED)
          return FALSE;
        if (retcode == JPEG_REACHED_EOI)
          break;
        /* Advance progress counter if appropriate */
        if (cinfo->progress != NULL &&
            (retcode == JPEG_ROW_COMPLETED || retcode == JPEG_REACHED_SOS)) {
          if (++cinfo->progress->pass_counter >= cinfo->progress->pass_limit) {
            /* jdmaster underestimated number of scans; ratchet up one scan */
            cinfo->progress->pass_limit += (long) cinfo->total_iMCU_rows;
          }
        }
      }
    }
  }
} else
  ERREXIT(cinfo, JERR_NOT_COMPILED);
#endif /* D_MULTISCAN_FILES_SUPPORTED */
}
cinfo->output_scan_number = cinfo->input_scan_number;
} else if (cinfo->global_state != DSTATE_PRESCAN)
  ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
}

```

```

/* Perform any dummy output passes, and set up for the final pass */
return output_pass_setup(cinfo);
}

/*
 * Set up for an output pass, and perform any dummy pass(es) needed.
 * Common subroutine for jpeg_start_decompress and jpeg_start_output.
 * Entry: global_state = DSTATE_PRESCAN only if previously suspended.
 * Exit: If done, returns TRUE and sets global_state for proper output mode.
 *       If suspended, returns FALSE and sets global_state = DSTATE_PRESCAN.
 */

LOCAL(boolean)
output_pass_setup (j_decompress_ptr cinfo)
{
    if (cinfo->global_state != DSTATE_PRESCAN) {
        /* First call: do pass setup */
        (*cinfo->master->prepare_for_output_pass) (cinfo);
        cinfo->output_scanline = 0;
        cinfo->global_state = DSTATE_PRESCAN;
    }
    /* Loop over any required dummy passes */
    while (cinfo->master->is_dummy_pass) {
#ifdef QUANT_2PASS_SUPPORTED
        /* Crank through the dummy pass */
        while (cinfo->output_scanline < cinfo->output_height) {
            JDIMENSION last_scanline;
            /* Call progress monitor hook if present */
            if (cinfo->progress != NULL) {
                cinfo->progress->pass_counter = (long) cinfo->output_scanline;
                cinfo->progress->pass_limit = (long) cinfo->output_height;
                (*cinfo->progress->progress_monitor) ((j_common_ptr) cinfo);
            }
            /* Process some data */
            last_scanline = cinfo->output_scanline;
            (*cinfo->main->process_data) (cinfo, (JSAMPARRAY) NULL,
                                         &cinfo->output_scanline, (JDIMENSION) 0);
            if (cinfo->output_scanline == last_scanline)
                return FALSE; /* No progress made, must suspend */
        }
        /* Finish up dummy pass, and set up for another one */
        (*cinfo->master->finish_output_pass) (cinfo);
        (*cinfo->master->prepare_for_output_pass) (cinfo);
        cinfo->output_scanline = 0;
    #else
        ERREXIT(cinfo, JERR_NOT_COMPILED);
    #endif /* QUANT_2PASS_SUPPORTED */
    }
    /* Ready for application to drive output pass through
     * jpeg_read_scanlines or jpeg_read_raw_data.
     */
    cinfo->global_state = cinfo->raw_data_out ? DSTATE_RAW_OK : DSTATE_SCANNING;
    return TRUE;
}

/*
 * Read some scanlines of data from the JPEG decompressor.
 *
 * The return value will be the number of lines actually read.
 * This may be less than the number requested in several cases,
 * including bottom of image, data source suspension, and operating
 * modes that emit multiple scanlines at a time.
 *
 * Note: we warn about excess calls to jpeg_read_scanlines() since
 * this likely signals an application programmer error. However,
 * an oversize buffer (max_lines > scanlines remaining) is not an error.
 */

GLOBAL(JDIMENSION)
jpeg_read_scanlines (j_decompress_ptr cinfo, JSAMPARRAY scanlines,
                    JDIMENSION max_lines)
{
    JDIMENSION row_ctr;

    if (cinfo->global_state != DSTATE_SCANNING)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
    if (cinfo->output_scanline >= cinfo->output_height) {
        WARNMS(cinfo, JWRN_TOO_MUCH_DATA);
    }
}

```

```

    return 0;
}

/* Call progress monitor hook if present */
if (cinfo->progress != NULL) {
    cinfo->progress->pass_counter = (long) cinfo->output_scanline;
    cinfo->progress->pass_limit = (long) cinfo->output_height;
    (*cinfo->progress->progress_monitor) ((j_common_ptr) cinfo);
}

/* Process some data */
row_ctr = 0;
(*cinfo->main->process_data) (cinfo, scanlines, &row_ctr, max_lines);
cinfo->output_scanline += row_ctr;
return row_ctr;
}

/*
 * Alternate entry point to read raw data.
 * Processes exactly one iMCU row per call, unless suspended.
 */

GLOBAL(JDIMENSION)
jpeg_read_raw_data (j_decompress_ptr cinfo, JSAMPIMAGE data,
                    JDIMENSION max_lines)
{
    JDIMENSION lines_per_iMCU_row;

    if (cinfo->global_state != DSTATE_RAW_OK)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
    if (cinfo->output_scanline >= cinfo->output_height) {
        WARNMS(cinfo, JWRN_TOO_MUCH_DATA);
        return 0;
    }

    /* Call progress monitor hook if present */
    if (cinfo->progress != NULL) {
        cinfo->progress->pass_counter = (long) cinfo->output_scanline;
        cinfo->progress->pass_limit = (long) cinfo->output_height;
        (*cinfo->progress->progress_monitor) ((j_common_ptr) cinfo);
    }

    /* Verify that at least one iMCU row can be returned. */
    lines_per_iMCU_row = cinfo->max_v_samp_factor * cinfo->min_DCT_scaled_size;
    if (max_lines < lines_per_iMCU_row)
        ERREXIT(cinfo, JERR_BUFFER_SIZE);

    /* Decompress directly into user's buffer. */
    if (! (*cinfo->coef->decompress_data) (cinfo, data))
        return 0; /* suspension forced, can do nothing more */

    /* OK, we processed one iMCU row. */
    cinfo->output_scanline += lines_per_iMCU_row;
    return lines_per_iMCU_row;
}

/* Additional entry points for buffered-image mode. */
#ifdef D_MULTISCAN_FILES_SUPPORTED

/*
 * Initialize for an output pass in buffered-image mode.
 */

GLOBAL(boolean)
jpeg_start_output (j_decompress_ptr cinfo, int scan_number)
{
    if (cinfo->global_state != DSTATE_BUFIMAGE &&
        cinfo->global_state != DSTATE_PRESCAN)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
    /* Limit scan number to valid range */
    if (scan_number <= 0)
        scan_number = 1;
    if (cinfo->inputctl->eoi_reached &&
        scan_number > cinfo->input_scan_number)
        scan_number = cinfo->input_scan_number;
    cinfo->output_scan_number = scan_number;
    /* Perform any dummy output passes, and set up for the real pass */

```

```

    return output_pass_setup(cinfo);
}

/*
 * Finish up after an output pass in buffered-image mode.
 *
 * Returns FALSE if suspended. The return value need be inspected only if
 * a suspending data source is used.
 */

GLOBAL(boolean)
jpeg_finish_output (j_decompress_ptr cinfo)
{
    if ((cinfo->global_state == DSTATE_SCANNING ||
        cinfo->global_state == DSTATE_RAW_OK) && cinfo->buffered_image) {
        /* Terminate this pass. */
        /* We do not require the whole pass to have been completed. */
        (*cinfo->master->finish_output_pass) (cinfo);
        cinfo->global_state = DSTATE_BUFPOST;
    } else if (cinfo->global_state != DSTATE_BUFPOST) {
        /* BUFPOST = repeat call after a suspension, anything else is error */
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
    }
    /* Read markers looking for SOS or EOI */
    while (cinfo->input_scan_number <= cinfo->output_scan_number &&
        ! cinfo->inputctl->eoi_reached) {
        if ((*cinfo->inputctl->consume_input) (cinfo) == JPEG_SUSPENDED)
            return FALSE; /* Suspend, come back later */
    }
    cinfo->global_state = DSTATE_BUFIMAGE;
    return TRUE;
}
#endif /* D_MULTISCAN_FILES_SUPPORTED */

```



```

/*
 * jdatadst.c
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains compression data destination routines for the case of
 * emitting JPEG data to a file (or any stdio stream). While these routines
 * are sufficient for most applications, some will want to use a different
 * destination manager.
 * IMPORTANT: we assume that fwrite() will correctly transcribe an array of
 * JOCTETs into 8-bit-wide elements on external storage. If char is wider
 * than 8 bits on your machine, you may need to do some tweaking.
 */

/* this is not a core library module, so it doesn't define JPEG_INTERNALS */
#include "jinclude.h"
#include "jpeglib.h"
#include "jerror.h"

/* ### a couple of new definitions to help differentiate between FILE and buffer I/O */
#define FILE_OUTPUT 1
#define BUFFER_OUTPUT 2

/* Expanded data destination object for stdio output */

typedef struct {
  struct jpeg_destination_mgr pub; /* public fields */

  FILE * outfile; /* target stream */
  JOCTET **outbuffer; /* /* target buffer */
  JOCTET * buffer; /* start of buffer */
  int n_BufferFlag; /* ### My addition. Flag determines whether to process FILE * or JOCTET * */
} my_destination_mgr;

typedef my_destination_mgr * my_dest_ptr;

#define OUTPUT_BUF_SIZE 4096 /* choose an efficiently fwrite'able size */

/* ### Imtiaz: Stitching routine to patch linked list into a single stream of JOCTET */
/* To store the stuff (JOCTET stream) in jpeg_destination_mgr.outbuffer; */

GLOBAL(void)
stitch_list(j_compress_ptr cinfo)
{
  int i;
  int chunk, count=0;
  buffer_list* list, *temp;

  list=cinfo->dest->head_ptr;
  chunk=(int)cinfo->index;

  /* Allocating memory. */

  cinfo->dest->outbuffer=(JOCTET *)malloc(cinfo->index*sizeof(JOCTET));
  if (cinfo->dest->outbuffer==NULL)
  {
    fprintf(stderr,"Memory Allocation Error\n");
    exit(1);
  }

  while (list->next!=NULL)
  {
    for(i=0;i<OUTPUT_BUF_SIZE;i++)
    {
      cinfo->dest->outbuffer[count]=list->buffer[i];
      count++;
    }

    temp=list;
    list=list->next;

    free(temp->buffer);
    free(temp);
  }
}

```

```

    chunk=chunk-(int)OUTPUT_BUF_SIZE;
}
/* the rest */
for(i=0;i<chunk;i++)
{
    cinfo->dest->outbuffer[count]=list->buffer[i];
    count++;
}

free(list->buffer);
free(list);
}

/*
 * Initialize destination --- called by jpeg_start_compress
 * before any data is actually written.
 */

METHODDEF(void)
init_destination (j_compress_ptr cinfo)
{
    my_dest_ptr dest = (my_dest_ptr) cinfo->dest;

    /* Allocate the output buffer --- it will be released when done with image */
    dest->buffer = (JOCTET *)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            OUTPUT_BUF_SIZE * SIZEOF(JOCTET));

    dest->pub.next_output_byte = dest->buffer;
    dest->pub.free_in_buffer = OUTPUT_BUF_SIZE;
}

/*
 * Empty the output buffer --- called whenever buffer fills up.
 *
 * In typical applications, this should write the entire output buffer
 * (ignoring the current state of next_output_byte & free_in_buffer),
 * reset the pointer & count to the start of the buffer, and return TRUE
 * indicating that the buffer has been dumped.
 *
 * In applications that need to be able to suspend compression due to output
 * overrun, a FALSE return indicates that the buffer cannot be emptied now.
 * In this situation, the compressor will return to its caller (possibly with
 * an indication that it has not accepted all the supplied scanlines). The
 * application should resume compression after it has made more room in the
 * output buffer. Note that there are substantial restrictions on the use of
 * suspension --- see the documentation.
 *
 * When suspending, the compressor will back up to a convenient restart point
 * (typically the start of the current MCU). next_output_byte & free_in_buffer
 * indicate where the restart point will be if the current call returns FALSE.
 * Data beyond this point will be regenerated after resumption, so do not
 * write it out when emptying the buffer externally.
 */

METHODDEF(boolean)
empty_output_buffer (j_compress_ptr cinfo)
{
    int i, index, increment;
    buffer_list *bl;

    my_dest_ptr dest = (my_dest_ptr) cinfo->dest;

    if (dest->n_BufferFlag==FILE_OUTPUT)
    {
        if (JFWRITE(dest->outfile, dest->buffer, OUTPUT_BUF_SIZE) !=
            (size_t) OUTPUT_BUF_SIZE)
            ERREXIT(cinfo, JERR_FILE_WRITE);
    }
    if (dest->n_BufferFlag==BUFFER_OUTPUT)
    {
        /* ### Imtiaz: First thing we know is that this part deals with a fixed length dump. */

```

```

bl=(buffer_list *)malloc(sizeof(buffer_list));
if (bl==NULL)
{
    fprintf(stderr,"Memory Allocation Error\n");
    exit(1);
}

bl->buffer=(JOCTET *)malloc(OUTPUT_BUF_SIZE*sizeof(JOCTET));

bl->next=NULL;

/* For the first time we need to set the head_ptr and current_ptr in jpeg_destination_mgr accordingl
y. Yes! I changed that too. */
if (cinfo->dest->head_ptr==NULL)
{
    cinfo->dest->current_ptr=bl;
    cinfo->dest->head_ptr=cinfo->dest->current_ptr;
}
else
{
    cinfo->dest->current_ptr->next=bl;
    cinfo->dest->current_ptr=bl;
}

index=cinfo->index;

increment=OUTPUT_BUF_SIZE;

for (i=0;i<OUTPUT_BUF_SIZE;i++)
    cinfo->dest->current_ptr->buffer[i]=dest->buffer[i];

cinfo->index=index+(int)OUTPUT_BUF_SIZE;

dest->pub.next_output_byte = dest->buffer;
dest->pub.free_in_buffer = OUTPUT_BUF_SIZE;

return TRUE;
}

Terminate destination --- called by jpeg_finish_compress
after all data has been written. Usually needs to flush buffer.

NB: *not* called by jpeg_abort or jpeg_destroy; surrounding
application must deal with any cleanup that should happen even
for error exit.
*/

```

```

METHODDEF(void)
term_destination (j_compress_ptr cinfo)
{
    int i,index;
    buffer_list *bl;

    my_dest_ptr dest = (my_dest_ptr) cinfo->dest;
    size_t datacount = OUTPUT_BUF_SIZE - dest->pub.free_in_buffer;

    /* Write any data remaining in the buffer */
    if (datacount > 0) {
        if (dest->n_BufferFlag==FILE_OUTPUT)
        {
            if (JFWRITE(dest->outfile, dest->buffer, datacount) != datacount)
                ERREXIT(cinfo, JERR_FILE_WRITE);

            fflush(dest->outfile);

            /* Make sure we wrote the output file OK */
            if (ferror(dest->outfile))
                ERREXIT(cinfo, JERR_FILE_WRITE);
        }
        if (dest->n_BufferFlag==BUFFER_OUTPUT)
        {

```

```

/* ### Imtiaz: First thing we know is that this part deals with a fixed length dump. */

bl=(buffer_list *)malloc(sizeof(buffer_list));
if (bl==NULL)
{
    fprintf(stderr, "Memory Allocation Error\n");
    exit(1);
}

bl->buffer=(JOCTET *)malloc(datacount*sizeof(JOCTET));

bl->next=NULL;

/* For the first time we need to set the head_ptr and current_ptr in jpeg_destination_mgr accordingly. Yes! I changed that too. */
if (cinfo->dest->head_ptr==NULL)
{
    cinfo->dest->current_ptr=bl;
    cinfo->dest->head_ptr=cinfo->dest->current_ptr;
}
else
{
    cinfo->dest->current_ptr->next=bl;
    cinfo->dest->current_ptr=bl;
}

index=cinfo->index;

for (i=0;i<(int)datacount;i++)
    cinfo->dest->current_ptr->buffer[i]=dest->buffer[i];

cinfo->index=index+datacount;

/* The end... Now to stitch. :) */
stitch_list(cinfo);
}

/*
 * Prepare for output to a stdio stream.
 * The caller must have already opened the stream, and is responsible
 * for closing it after finishing compression.
 */

GLOBAL(void)
jpeg_stdio_dest (j_compress_ptr cinfo, FILE * outfile)
{
    my_dest_ptr dest;

    /* The destination object is made permanent so that multiple JPEG images
     * can be written to the same file without re-executing jpeg_stdio_dest.
     * This makes it dangerous to use this manager and a different destination
     * manager serially with the same JPEG object, because their private object
     * sizes may be different.  Caveat programmer.
     */
    if (cinfo->dest == NULL) { /* first time for this JPEG object? */
        cinfo->dest = (struct jpeg_destination_mgr *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_PERMANENT,
                sizeof(my_destination_mgr));
    }

    dest = (my_dest_ptr) cinfo->dest;

    /* ### Setting flag to process FILE output */
    dest->n_BufferFlag=FILE_OUTPUT;

    dest->pub.init_destination = init_destination;
    dest->pub.empty_output_buffer = empty_output_buffer;
    dest->pub.term_destination = term_destination;
    dest->outfile = outfile;
}

```

```
}
```

```
GLOBAL(void)
jpeg_buffer_dest (j_compress_ptr cinfo)
{
    my_dest_ptr dest;

    /* The destination object is made permanent so that multiple JPEG images
     * can be written to the same file without re-executing jpeg_stdio_dest.
     * This makes it dangerous to use this manager and a different destination
     * manager serially with the same JPEG object, because their private object
     * sizes may be different.  Caveat programmer.
     */
    if (cinfo->dest == NULL) { /* first time for this JPEG object? */
        cinfo->dest = (struct jpeg_destination_mgr *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_PERMANENT,
                sizeof(my_destination_mgr));
    }

    dest = (my_dest_ptr) cinfo->dest;

    /* ### Intiaz: Setting flag to process BUFFER output */
    dest->n_BufferFlag=BUFFER_OUTPUT;

    dest->pub.init_destination = init_destination;
    dest->pub.empty_output_buffer = empty_output_buffer;
    dest->pub.term_destination = term_destination;
    /* ### cinfo->dest->outbuffer = buff; */

    cinfo->dest->head_ptr=NULL;
    cinfo->dest->current_ptr=NULL;
}
```

```

/*
 * jdatasrc.c
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains decompression data source routines for the case of
 * reading JPEG data from a file (or any stdio stream). While these routines
 * are sufficient for most applications, some will want to use a different
 * source manager.
 * IMPORTANT: we assume that fread() will correctly transcribe an array of
 * JOCTETs from 8-bit-wide elements on external storage. If char is wider
 * than 8 bits on your machine, you may need to do some tweaking.
 */

/* this is not a core library module, so it doesn't define JPEG_INTERNALS */
#include "jinclude.h"
#include "jpeglib.h"
#include "jerror.h"

#define FILE_OUTPUT 1 /* ### Imtiaz: I put this here. */
#define BUFFER_OUTPUT 2

/* Expanded data source object for stdio input */
typedef struct {
  struct jpeg_source_mgr pub; /* public fields */

  FILE * infile; /* source stream */
  JOCTET * buffer; /* start of buffer */
  boolean start_of_file; /* have we gotten any data yet? */
  long count_track;

  int n_BufferFlag; /* ### Imtiaz: My addition. Flag determines whether to process FILE * or JOCTET */
} my_source_mgr;

typedef my_source_mgr * my_src_ptr;

#define INPUT_BUF_SIZE 4096 /* choose an efficiently fread'able size */

/*
 * Initialize source --- called by jpeg_read_header
 * before any data is actually read.
 */
METHODDEF(void)
init_source (j_decompress_ptr cinfo)
{
  my_src_ptr src = (my_src_ptr) cinfo->src;

  /* We reset the empty-input-file flag for each image,
   * but we don't clear the input buffer.
   * This is correct behavior for reading a series of images from one source.
   */
  src->start_of_file = TRUE;
}

/*
 * Fill the input buffer --- called whenever buffer is emptied.
 *
 * In typical applications, this should read fresh data into the buffer
 * (ignoring the current state of next_input_byte & bytes_in_buffer),
 * reset the pointer & count to the start of the buffer, and return TRUE
 * indicating that the buffer has been reloaded. It is not necessary to
 * fill the buffer entirely, only to obtain at least one more byte.
 *
 * There is no such thing as an EOF return. If the end of the file has been
 * reached, the routine has a choice of ERREXIT() or inserting fake data into
 * the buffer. In most cases, generating a warning message and inserting a
 * fake EOI marker is the best course of action --- this will allow the
 * decompressor to output however much of the image is there. However,
 * the resulting error message is misleading if the real problem is an empty
 * input file, so we handle that case specially.
 */

```

```

* In applications that need to be able to suspend compression due to input
* not being available yet, a FALSE return indicates that no more data can be
* obtained right now, but more may be forthcoming later. In this situation,
* the decompressor will return to its caller (with an indication of the
* number of scanlines it has read, if any). The application should resume
* decompression after it has loaded more data into the input buffer. Note
* that there are substantial restrictions on the use of suspension --- see
* the documentation.
*
* When suspending, the decompressor will back up to a convenient restart point
* (typically the start of the current MCU). next_input_byte & bytes_in_buffer
* indicate where the restart point will be if the current call returns FALSE.
* Data beyond this point must be rescanned after resumption, so move it to
* the front of the buffer rather than discarding it.
*/

```

```

METHODDEF(boolean)
fill_input_buffer (j_decompress_ptr cinfo)
{
    long count;
    static int p;

    my_src_ptr src = (my_src_ptr) cinfo->src;
    size_t nbytes;

    if (src->n_BufferFlag==FILE_OUTPUT)
    {
        nbytes = JFREAD(src->infile, src->buffer, INPUT_BUF_SIZE);
    }
    else
    {
        count=0;
        while ((p<cinfo->src->buffer_length)&&(count<INPUT_BUF_SIZE))
        {
            src->buffer[count]= cinfo->src->inbuffer[0];
            count++;

            cinfo->src->inbuffer++;
            p++;
        }
        nbytes=(size_t)count;
    }
    if (nbytes <= 0) {
        if (src->n_BufferFlag==FILE_OUTPUT)
        {
            if (src->start_of_file) /* Treat empty input file as fatal error */
                ERREXIT(cinfo, JERR_INPUT_EMPTY);
            WARNMS(cinfo, JWRN_JPEG_EOF);
        }
        else
        { /* if dest->n_BufferFlag==BUFFER_OUTPUT */
            fprintf(stderr, "Buffer Empty\n");
            fprintf(stderr, "Using Fake EOI marker.. \n");
        }
        /* Insert a fake EOI marker */
        src->buffer[0] = (JOCTET) 0xFF;
        src->buffer[1] = (JOCTET) JPEG_EOI;
        nbytes = 2;
    }

    src->pub.next_input_byte = src->buffer;
    src->pub.bytes_in_buffer = nbytes;
    src->start_of_file = FALSE;

    return TRUE;
}

/*
* Skip data --- used to skip over a potentially large amount of
* uninteresting data (such as an APPn marker).
*
* Writers of suspendable-input applications must note that skip_input_data
* is not granted the right to give a suspension return. If the skip extends

```

```

* beyond the data currently in the buffer, the buffer can be marked empty so
* that the next read will cause a fill_input_buffer call that can suspend.
* Arranging for additional bytes to be discarded before reloading the input
* buffer is the application writer's problem.
*/

```

```

METHODDEF(void)
skip_input_data (j_decompress_ptr cinfo, long num_bytes)
{
    my_src_ptr src = (my_src_ptr) cinfo->src;

    /* Just a dumb implementation for now. Could use fseek() except
     * it doesn't work on pipes. Not clear that being smart is worth
     * any trouble anyway --- large skips are infrequent.
     */
    if (num_bytes > 0) {
        while (num_bytes > (long) src->pub.bytes_in_buffer) {
            num_bytes -= (long) src->pub.bytes_in_buffer;
            (void) fill_input_buffer(cinfo);
            /* note we assume that fill_input_buffer will never return FALSE,
             * so suspension need not be handled.
             */
        }
        src->pub.next_input_byte += (size_t) num_bytes;
        src->pub.bytes_in_buffer -= (size_t) num_bytes;
    }
}

```

```

/*
 * An additional method that can be provided by data source modules is the
 * resync_to_restart method for error recovery in the presence of RST markers.
 * For the moment, this source module just uses the default resync method
 * provided by the JPEG library. That method assumes that no backtracking
 * is possible.
 */

```

```

/*
 * Terminate source --- called by jpeg_finish_decompress
 * after all data has been read. Often a no-op.
 * NB: *not* called by jpeg_abort or jpeg_destroy; surrounding
 * application must deal with any cleanup that should happen even
 * for error exit.
 */

```

```

METHODDEF(void)
term_source (j_decompress_ptr cinfo)
{
    /* no work necessary here */
}

```

```

/*
 * Prepare for input from a stdio stream.
 * The caller must have already opened the stream, and is responsible
 * for closing it after finishing decompression.
 */

```

```

GLOBAL(void)
jpeg_stdio_src (j_decompress_ptr cinfo, FILE * infile)
{
    my_src_ptr src;

    /* The source object and input buffer are made permanent so that a series
     * of JPEG images can be read from the same file by calling jpeg_stdio_src
     * only before the first one. (If we discarded the buffer at the end of
     * one image, we'd likely lose the start of the next one.)
     * This makes it unsafe to use this manager and a different source
     * manager serially with the same JPEG object. Caveat programmer.
     */
    if (cinfo->src == NULL) { /* first time for this JPEG object? */
        cinfo->src = (struct jpeg_source_mgr *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_PERMANENT,
                                      SIZEOF(my_source_mgr));
        src = (my_src_ptr) cinfo->src;
        src->buffer = (JOCTET *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_PERMANENT,
                                      INPUT_BUF_SIZE * SIZEOF(JOCTET));
    }
}

```



```

    }

    src = (my_src_ptr) cinfo->src;
    src->pub.init_source = init_source;
    src->pub.fill_input_buffer = fill_input_buffer;
    src->pub.skip_input_data = skip_input_data;
    src->pub.resync_to_restart = jpeg_resync_to_restart; /* use default method */
    src->pub.term_source = term_source;
    src->infile = infile;
    src->pub.bytes_in_buffer = 0; /* forces fill_input_buffer on first read */
    src->pub.next_input_byte = NULL; /* until buffer loaded */
}

```

```

GLOBAL(void)
jpeg_buffer_src (j_decompress_ptr cinfo)
{
    my_src_ptr src;

    /* The source object and input buffer are made permanent so that a series
     * of JPEG images can be read from the same file by calling jpeg_stdio_src
     * only before the first one. (If we discarded the buffer at the end of
     * one image, we'd likely lose the start of the next one.)
     * This makes it unsafe to use this manager and a different source
     * manager serially with the same JPEG object. Caveat programmer.
     */
    if (cinfo->src == NULL) { /* first time for this JPEG object? */
        cinfo->src = (struct jpeg_source_mgr *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_PERMANENT,
                                       SIZEOF(my_source_mgr));
        src = (my_src_ptr) cinfo->src;
        src->buffer = (JOCTET *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_PERMANENT,
                                       INPUT_BUF_SIZE * SIZEOF(JOCTET));
    }

    src = (my_src_ptr) cinfo->src;

    /* ### Imtiaz: Setting flag to process BUFFER output */
    src->n_BufferFlag=BUFFER_OUTPUT;

    src->pub.init_source = init_source;
    src->pub.fill_input_buffer = fill_input_buffer;
    src->pub.skip_input_data = skip_input_data;
    src->pub.resync_to_restart = jpeg_resync_to_restart; /* use default method */
    src->pub.term_source = term_source;

    src->pub.bytes_in_buffer = 0; /* forces fill_input_buffer on first read */
    src->pub.next_input_byte = NULL; /* until buffer loaded */
}

```

```

/*
 * jdcoefct.c
 *
 * Copyright (C) 1994-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains the coefficient buffer controller for decompression.
 * This controller is the top level of the JPEG decompressor proper.
 * The coefficient buffer lies between entropy decoding and inverse-DCT steps.
 *
 * In buffered-image mode, this controller is the interface between
 * input-oriented processing and output-oriented processing.
 * Also, the input side (only) is used when reading a file for transcoding.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/* Block smoothing is only applicable for progressive JPEG, so: */
#ifdef D_PROGRESSIVE_SUPPORTED
#undef BLOCK_SMOOTHING_SUPPORTED
#endif

/* Private buffer controller object */
typedef struct {
  struct jpeg_d_coef_controller pub; /* public fields */

  /* These variables keep track of the current location of the input side. */
  /* cinfo->input_iMCU_row is also used for this. */
  int DIMENSION MCU_ctr; /* counts MCUs processed in current row */
  int MCU_vert_offset; /* counts MCU rows within iMCU row */
  int MCU_rows_per_iMCU_row; /* number of such rows needed */

  /* The output side's location is represented by cinfo->output_iMCU_row. */

  /* In single-pass modes, it's sufficient to buffer just one MCU.
   * We allocate a workspace of D_MAX_BLOCKS_IN_MCU coefficient blocks,
   * and let the entropy decoder write into that workspace each time.
   * (On 80x86, the workspace is FAR even though it's not really very big;
   * this is to keep the module interfaces unchanged when a large coefficient
   * buffer is necessary.)
   * In multi-pass modes, this array points to the current MCU's blocks
   * within the virtual arrays; it is used only by the input side.
   */
  JBLOCKROW MCU_buffer[D_MAX_BLOCKS_IN_MCU];

#ifdef D_MULTISCAN_FILES_SUPPORTED
  /* In multi-pass modes, we need a virtual block array for each component. */
  jvirt_barray_ptr whole_image[MAX_COMPONENTS];
#endif

#ifdef BLOCK_SMOOTHING_SUPPORTED
  /* When doing block smoothing, we latch coefficient Al values here */
  int * coef_bits_latch;
#define SAVED_COEFS 6 /* we save coef_bits[0..5] */
#endif
} my_coef_controller;

typedef my_coef_controller * my_coef_ptr;

/* Forward declarations */
METHODDEF(int) decompress_onepass
  JPP((j_decompress_ptr cinfo, JSAMPIMAGE output_buf));
#ifdef D_MULTISCAN_FILES_SUPPORTED
METHODDEF(int) decompress_data
  JPP((j_decompress_ptr cinfo, JSAMPIMAGE output_buf));
#endif
#ifdef BLOCK_SMOOTHING_SUPPORTED
LOCAL(boolean) smoothing_ok JPP((j_decompress_ptr cinfo));
METHODDEF(int) decompress_smooth_data
  JPP((j_decompress_ptr cinfo, JSAMPIMAGE output_buf));
#endif

LOCAL(void)
start_iMCU_row (j_decompress_ptr cinfo)
/* Reset within-iMCU-row counters for a new row (input side) */

```

```

{
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;

    /* In an interleaved scan, an MCU row is the same as an iMCU row.
     * In a noninterleaved scan, an iMCU row has v_samp_factor MCU rows.
     * But at the bottom of the image, process only what's left.
     */
    if (cinfo->comps_in_scan > 1) {
        coef->MCU_rows_per_iMCU_row = 1;
    } else {
        if (cinfo->input_iMCU_row < (cinfo->total_iMCU_rows-1))
            coef->MCU_rows_per_iMCU_row = cinfo->cur_comp_info[0]->v_samp_factor;
        else
            coef->MCU_rows_per_iMCU_row = cinfo->cur_comp_info[0]->last_row_height;
    }

    coef->MCU_ctr = 0;
    coef->MCU_vert_offset = 0;
}

/*
 * Initialize for an input processing pass.
 */

METHODDEF(void)
start_input_pass (j_decompress_ptr cinfo)
{
    cinfo->input_iMCU_row = 0;
    start_iMCU_row(cinfo);
}

/*
 * Initialize for an output processing pass.
 */

METHODDEF(void)
start_output_pass (j_decompress_ptr cinfo)
{
#ifdef BLOCK_SMOOTHING_SUPPORTED
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;

    /* If multipass, check to see whether to use block smoothing on this pass */
    if (coef->pub.coef_arrays != NULL) {
        if (cinfo->do_block_smoothing && smoothing_ok(cinfo))
            coef->pub.decompress_data = decompress_smooth_data;
        else
            coef->pub.decompress_data = decompress_data;
    }
#endif
    cinfo->output_iMCU_row = 0;
}

/*
 * Decompress and return some data in the single-pass case.
 * Always attempts to emit one fully interleaved MCU row ("iMCU" row).
 * Input and output must run in lockstep since we have only a one-MCU buffer.
 * Return value is JPEG_ROW_COMPLETED, JPEG_SCAN_COMPLETED, or JPEG_SUSPENDED.
 *
 * NB: output_buf contains a plane for each component in image,
 * which we index according to the component's SOF position.
 */

METHODDEF(int)
decompress_onepass (j_decompress_ptr cinfo, JSAMPIMAGE output_buf)
{
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;
    JDIMENSION MCU_col_num; /* index of current MCU within row */
    JDIMENSION last_MCU_col = cinfo->MCUs_per_row - 1;
    JDIMENSION last_iMCU_row = cinfo->total_iMCU_rows - 1;
    int blkn, ci, xindex, yindex, yoffset, useful_width;
    JSAMPARRAY output_ptr;
    JDIMENSION start_col;
    jpeg_component_info *comp_ptr;
    inverse_DCT_method_ptr inverse_DCT;

    /* Loop to process as much as one whole iMCU row */
    for (yoffset = coef->MCU_vert_offset; yoffset < coef->MCU_rows_per_iMCU_row;

```

```

        yoffset++) {
    for (MCU_col_num = coef->MCU_ctr; MCU_col_num <= last_MCU_col;
        MCU_col_num++) {
        /* Try to fetch an MCU. Entropy decoder expects buffer to be zeroed. */
        jzero_far((void *) coef->MCU_buffer[0],
            (size_t) (cinfo->blocks_in_MCU * SIZEOF(JBLOCK)));
        if (!(*cinfo->entropy->decode_mcu) (cinfo, coef->MCU_buffer)) {
            /* Suspension forced; update state counters and exit */
            coef->MCU_vert_offset = yoffset;
            coef->MCU_ctr = MCU_col_num;
            return JPEG_SUSPENDED;
        }
        /* Determine where data should go in output_buf and do the IDCT thing.
         * We skip dummy blocks at the right and bottom edges (but blk_n gets
         * incremented past them!). Note the inner loop relies on having
         * allocated the MCU_buffer[] blocks sequentially.
         */
        blk_n = 0; /* index of current DCT block within MCU */
        for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
            comp_ptr = cinfo->cur_comp_info[ci];
            /* Don't bother to IDCT an uninteresting component. */
            if (!comp_ptr->component_needed) {
                blk_n += comp_ptr->MCU_blocks;
                continue;
            }
            inverse_DCT = cinfo->idct->inverse_DCT[comp_ptr->component_index];
            useful_width = (MCU_col_num < last_MCU_col) ? comp_ptr->MCU_width
                : comp_ptr->last_col_width;
            output_ptr = output_buf[comp_ptr->component_index] +
                yoffset * comp_ptr->DCT_scaled_size;
            start_col = MCU_col_num * comp_ptr->MCU_sample_width;
            for (yindex = 0; yindex < comp_ptr->MCU_height; yindex++) {
                if (cinfo->input_iMCU_row < last_iMCU_row ||
                    yoffset+yindex < comp_ptr->last_row_height) {
                    output_col = start_col;
                    for (xindex = 0; xindex < useful_width; xindex++) {
                        (*inverse_DCT) (cinfo, comp_ptr,
                            (JCOEFPTR) coef->MCU_buffer[blk_n+xindex],
                            output_ptr, output_col);
                        output_col += comp_ptr->DCT_scaled_size;
                    }
                }
                blk_n += comp_ptr->MCU_width;
                output_ptr += comp_ptr->DCT_scaled_size;
            }
        }
        /* Completed an MCU row, but perhaps not an iMCU row */
        coef->MCU_ctr = 0;

        /* Completed the iMCU row, advance counters for next one */
        cinfo->output_iMCU_row++;
        if (++(cinfo->input_iMCU_row) < cinfo->total_iMCU_rows) {
            start_iMCU_row(cinfo);
            return JPEG_ROW_COMPLETED;
        }
        /* Completed the scan */
        (*cinfo->inputctl->finish_input_pass) (cinfo);
        return JPEG_SCAN_COMPLETED;
    }

    /*
     * Dummy consume-input routine for single-pass operation.
     */

METHODDEF(int)
dummy_consume_data (j_decompress_ptr cinfo)
{
    return JPEG_SUSPENDED; /* Always indicate nothing was done */
}

#ifdef D_MULTISCAN_FILES_SUPPORTED

/*
 * Consume input data and store it in the full-image coefficient buffer.
 * We read as much as one fully interleaved MCU row ("iMCU" row) per call,
 * ie, v_samp_factor block rows for each component in the scan.
 * Return value is JPEG_ROW_COMPLETED, JPEG_SCAN_COMPLETED, or JPEG_SUSPENDED.

```

```

*/

METHODDEF(int)
consume_data (j_decompress_ptr cinfo)
{
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;
    JDIMENSION MCU_col_num; /* index of current MCU within row */
    int blkn, ci, xindex, yindex, yoffset;
    JDIMENSION start_col;
    JBLOCKARRAY buffer[MAX_COMPS_IN_SCAN];
    JBLOCKROW buffer_ptr;
    jpeg_component_info *comp_ptr;

    /* Align the virtual buffers for the components used in this scan. */
    for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
        comp_ptr = cinfo->cur_comp_info[ci];
        buffer[ci] = (*cinfo->mem->access_virt_barray)
            ((j_common_ptr) cinfo, coef->whole_image[comp_ptr->component_index],
             cinfo->input_iMCU_row * comp_ptr->v_samp_factor,
             (JDIMENSION) comp_ptr->v_samp_factor, TRUE);
        /* Note: entropy decoder expects buffer to be zeroed,
         * but this is handled automatically by the memory manager
         * because we requested a pre-zeroed array.
         */
    }

    /* Loop to process one whole iMCU row */
    for (yoffset = coef->MCU_vert_offset; yoffset < coef->MCU_rows_per_iMCU_row;
         yoffset++) {
        for (MCU_col_num = coef->MCU_ctr; MCU_col_num < cinfo->MCUs_per_row;
             MCU_col_num++) {
            /* Construct list of pointers to DCT blocks belonging to this MCU */
            blkn = 0; /* index of current DCT block within MCU */
            for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
                comp_ptr = cinfo->cur_comp_info[ci];
                start_col = MCU_col_num * comp_ptr->MCU_width;
                for (yindex = 0; yindex < comp_ptr->MCU_height; yindex++) {
                    buffer_ptr = buffer[ci][yindex+yoffset] + start_col;
                    for (xindex = 0; xindex < comp_ptr->MCU_width; xindex++) {
                        coef->MCU_buffer[blkn++] = buffer_ptr++;
                    }
                }
            }
            /* Try to fetch the MCU. */
            if (!(*cinfo->entropy->decode_mcu) (cinfo, coef->MCU_buffer)) {
                /* Suspension forced; update state counters and exit */
                coef->MCU_vert_offset = yoffset;
                coef->MCU_ctr = MCU_col_num;
                return JPEG_SUSPENDED;
            }
        }
        /* Completed an MCU row, but perhaps not an iMCU row */
        coef->MCU_ctr = 0;
    }
    /* Completed the iMCU row, advance counters for next one */
    if (++(cinfo->input_iMCU_row) < cinfo->total_iMCU_rows) {
        start_iMCU_row(cinfo);
        return JPEG_ROW_COMPLETED;
    }
    /* Completed the scan */
    (*cinfo->inputctl->finish_input_pass) (cinfo);
    return JPEG_SCAN_COMPLETED;
}

/*
 * Decompress and return some data in the multi-pass case.
 * Always attempts to emit one fully interleaved MCU row ("iMCU" row).
 * Return value is JPEG_ROW_COMPLETED, JPEG_SCAN_COMPLETED, or JPEG_SUSPENDED.
 * NB: output_buf contains a plane for each component in image.
 */

```

```

METHODDEF(int)
decompress_data (j_decompress_ptr cinfo, JSAMPIMAGE output_buf)
{
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;
    JDIMENSION last_iMCU_row = cinfo->total_iMCU_rows - 1;
    JDIMENSION block_num;
    int ci, block_row, block_rows;

```

```

JBLOCKARRAY buffer;
JBLOCKROW buffer_ptr;
JSAMPARRAY output_ptr;
JDIMENSION output_col;
jpeg_component_info *comp_ptr;
inverse_DCT_method_ptr inverse_DCT;

/* Force some input to be done if we are getting ahead of the input. */
while (cinfo->input_scan_number < cinfo->output_scan_number ||
      (cinfo->input_scan_number == cinfo->output_scan_number &&
       cinfo->input_iMCU_row <= cinfo->output_iMCU_row)) {
    if ((*cinfo->inputctl->consume_input)(cinfo) == JPEG_SUSPENDED)
        return JPEG_SUSPENDED;
}

/* OK, output from the virtual arrays. */
for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
     ci++, comp_ptr++) {
    /* Don't bother to IDCT an uninteresting component. */
    if (!comp_ptr->component_needed)
        continue;
    /* Align the virtual buffer for this component. */
    buffer = (*cinfo->mem->access_virt_barray)
        ((j_common_ptr) cinfo, coef->whole_image[ci],
         cinfo->output_iMCU_row * comp_ptr->v_samp_factor,
         (JDIMENSION) comp_ptr->v_samp_factor, FALSE);
    /* Count non-dummy DCT block rows in this iMCU row. */
    if (cinfo->output_iMCU_row < last_iMCU_row)
        block_rows = comp_ptr->v_samp_factor;
    else {
        /* NB: can't use last_row_height here; it is input-side-dependent! */
        block_rows = (int) (comp_ptr->height_in_blocks % comp_ptr->v_samp_factor);
        if (block_rows == 0) block_rows = comp_ptr->v_samp_factor;
    }
    inverse_DCT = cinfo->idct->inverse_DCT[ci];
    output_ptr = output_buf[ci];
    /* Loop over all DCT blocks to be processed. */
    for (block_row = 0; block_row < block_rows; block_row++) {
        buffer_ptr = buffer[block_row];
        output_col = 0;
        for (block_num = 0; block_num < comp_ptr->width_in_blocks; block_num++) {
            (*inverse_DCT) (cinfo, comp_ptr, (JCOEFPTR) buffer_ptr,
                           output_ptr, output_col);
            buffer_ptr++;
            output_col += comp_ptr->DCT_scaled_size;
        }
        output_ptr += comp_ptr->DCT_scaled_size;
    }
    if (++(cinfo->output_iMCU_row) < cinfo->total_iMCU_rows)
        return JPEG_ROW_COMPLETED;
    return JPEG_SCAN_COMPLETED;
}

#endif /* D_MULTISCAN_FILES_SUPPORTED */

#ifdef BLOCK_SMOOTHING_SUPPORTED

/*
 * This code applies interblock smoothing as described by section K.8
 * of the JPEG standard: the first 5 AC coefficients are estimated from
 * the DC values of a DCT block and its 8 neighboring blocks.
 * We apply smoothing only for progressive JPEG decoding, and only if
 * the coefficients it can estimate are not yet known to full precision.
 */

/* Natural-order array positions of the first 5 zigzag-order coefficients */
#define Q01_POS 1
#define Q10_POS 8
#define Q20_POS 16
#define Q11_POS 9
#define Q02_POS 2

/*
 * Determine whether block smoothing is applicable and safe.
 * We also latch the current states of the coef_bits[] entries for the
 * AC coefficients; otherwise, if the input side of the decompressor
 * advances into a new scan, we might think the coefficients are known

```

```

* more accurately than they really are.
*/

```

```

LOCAL(boolean)
smoothing_ok (j_decompress_ptr cinfo)
{
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;
    boolean smoothing_useful = FALSE;
    int ci, coefi;
    jpeg_component_info *comp_ptr;
    JQUANT_TBL * qtable;
    int * coef_bits;
    int * coef_bits_latch;

    if (! cinfo->progressive_mode || cinfo->coef_bits == NULL)
        return FALSE;

    /* Allocate latch area if not already done */
    if (coef->coef_bits_latch == NULL)
        coef->coef_bits_latch = (int *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                cinfo->num_components *
                (SAVED_COEFS * SIZEOF(int)));
    coef_bits_latch = coef->coef_bits_latch;

    for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
        ci++, comp_ptr++) {
        /* All components' quantization values must already be latched. */
        if ((qtable = comp_ptr->quant_table) == NULL)
            return FALSE;
        /* Verify DC & first 5 AC quantizers are nonzero to avoid zero-divide. */
        if (qtable->quantval[0] == 0 ||
            qtable->quantval[Q01_POS] == 0 ||
            qtable->quantval[Q10_POS] == 0 ||
            qtable->quantval[Q20_POS] == 0 ||
            qtable->quantval[Q11_POS] == 0 ||
            qtable->quantval[Q02_POS] == 0)
            return FALSE;
        /* DC values must be at least partly known for all components. */
        coef_bits = cinfo->coef_bits[ci];
        if (coef_bits[0] < 0)
            return FALSE;
        /* Block smoothing is helpful if some AC coefficients remain inaccurate. */
        for (coefi = 1; coefi <= 5; coefi++) {
            coef_bits_latch[coefi] = coef_bits[coefi];
            if (coef_bits[coefi] != 0)
                smoothing_useful = TRUE;
        }
        coef_bits_latch += SAVED_COEFS;
    }
    return smoothing_useful;
}

```

```

/*
* Variant of decompress_data for use when doing block smoothing.
*/

```

```

METHODDEF(int)
decompress_smooth_data (j_decompress_ptr cinfo, JSAMPIMAGE output_buf)
{
    my_coef_ptr coef = (my_coef_ptr) cinfo->coef;
    JDIMENSION last_imcu_row = cinfo->total_imcu_rows - 1;
    JDIMENSION block_num, last_block_column;
    int ci, block_row, block_rows, access_rows;
    JBLOCKARRAY buffer;
    JBLOCKROW buffer_ptr, prev_block_row, next_block_row;
    JSAMPARRAY output_ptr;
    JDIMENSION output_col;
    jpeg_component_info *comp_ptr;
    inverse_DCT_method_ptr inverse_DCT;
    boolean first_row, last_row;
    JBLOCK workspace;
    int *coef_bits;
    JQUANT_TBL * quanttbl;
    INT32 Q00, Q01, Q02, Q10, Q11, Q20, num;
    int DC1, DC2, DC3, DC4, DC5, DC6, DC7, DC8, DC9;
    int A1, pred;
}

```

```

/* Force some input to be done if we are getting ahead of the input. */
while (cinfo->input_scan_number <= cinfo->output_scan_number &&
! cinfo->inputctl->eoi_reached) {
    if (cinfo->input_scan_number == cinfo->output_scan_number) {
        /* If input is working on current scan, we ordinarily want it to
         * have completed the current row. But if input scan is DC,
         * we want it to keep one row ahead so that next block row's DC
         * values are up to date.
         */
        JDIMENSION delta = (cinfo->Ss == 0) ? 1 : 0;
        if (cinfo->input_iMCU_row > cinfo->output_iMCU_row+delta)
            break;
    }
    if ((*cinfo->inputctl->consume_input)(cinfo) == JPEG_SUSPENDED)
        return JPEG_SUSPENDED;
}

/* OK, output from the virtual arrays. */
for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
     ci++, compptr++) {
    /* Don't bother to IDCT an uninteresting component. */
    if (! compptr->component_needed)
        continue;
    /* Count non-dummy DCT block rows in this iMCU row. */
    if (cinfo->output_iMCU_row < last_iMCU_row) {
        block_rows = compptr->v_samp_factor;
        access_rows = block_rows * 2; /* this and next iMCU row */
        last_row = FALSE;
    } else {
        /* NB: can't use last_row_height here; it is input-side-dependent! */
        block_rows = (int) (compptr->height_in_blocks % compptr->v_samp_factor);
        if (block_rows == 0) block_rows = compptr->v_samp_factor;
        access_rows = block_rows; /* this iMCU row only */
        last_row = TRUE;
    }
    /* Align the virtual buffer for this component. */
    if (cinfo->output_iMCU_row > 0) {
        access_rows += compptr->v_samp_factor; /* prior iMCU row too */
        buffer = (*cinfo->mem->access_virt_barray)
            ((j_common_ptr) cinfo, coef->whole_image[ci],
             (cinfo->output_iMCU_row - 1) * compptr->v_samp_factor,
             (JDIMENSION) access_rows, FALSE);
        buffer += compptr->v_samp_factor; /* point to current iMCU row */
        first_row = FALSE;
    } else {
        buffer = (*cinfo->mem->access_virt_barray)
            ((j_common_ptr) cinfo, coef->whole_image[ci],
             (JDIMENSION) 0, (JDIMENSION) access_rows, FALSE);
        first_row = TRUE;
    }
    /* Fetch component-dependent info */
    coef_bits = coef->coef_bits_latch + (ci * SAVED_COEFS);
    quanttbl = compptr->quant_table;
    Q00 = quanttbl->quantval[0];
    Q01 = quanttbl->quantval[Q01_POS];
    Q10 = quanttbl->quantval[Q10_POS];
    Q20 = quanttbl->quantval[Q20_POS];
    Q11 = quanttbl->quantval[Q11_POS];
    Q02 = quanttbl->quantval[Q02_POS];
    inverse_DCT = cinfo->idct->inverse_DCT[ci];
    output_ptr = output_buf[ci];
    /* Loop over all DCT blocks to be processed. */
    for (block_row = 0; block_row < block_rows; block_row++) {
        buffer_ptr = buffer[block_row];
        if (first_row && block_row == 0)
            prev_block_row = buffer_ptr;
        else
            prev_block_row = buffer[block_row-1];
        if (last_row && block_row == block_rows-1)
            next_block_row = buffer_ptr;
        else
            next_block_row = buffer[block_row+1];
        /* We fetch the surrounding DC values using a sliding-register approach.
         * Initialize all nine here so as to do the right thing on narrow pics.
         */
        DC1 = DC2 = DC3 = (int) prev_block_row[0][0];
        DC4 = DC5 = DC6 = (int) buffer_ptr[0][0];
        DC7 = DC8 = DC9 = (int) next_block_row[0][0];
        output_col = 0;
        last_block_column = compptr->width_in_blocks - 1;
    }
}

```



```

    for (block_num = 0; block_num <= last_block_column; block_num++) {
/* Fetch current DCT block into workspace so we can modify it. */
jcopy_block_row(buffer_ptr, (JBLOCKROW) workspace, (JDIMENSION) 1);
/* Update DC values */
if (block_num < last_block_column) {
    DC3 = (int) prev_block_row[1][0];
    DC6 = (int) buffer_ptr[1][0];
    DC9 = (int) next_block_row[1][0];
}
/* Compute coefficient estimates per K.8.
 * An estimate is applied only if coefficient is still zero,
 * and is not known to be fully accurate.
 */
/* AC01 */
if ((Al=coef_bits[1]) != 0 && workspace[1] == 0) {
    num = 36 * Q00 * (DC4 - DC6);
    if (num >= 0) {
        pred = (int) (((Q01<<7) + num) / (Q01<<8));
        if (Al > 0 && pred >= (1<<Al))
            pred = (1<<Al)-1;
    } else {
        pred = (int) (((Q01<<7) - num) / (Q01<<8));
        if (Al > 0 && pred >= (1<<Al))
            pred = (1<<Al)-1;
        pred = -pred;
    }
    workspace[1] = (JCOEF) pred;
}
/* AC10 */
if ((Al=coef_bits[2]) != 0 && workspace[8] == 0) {
    num = 36 * Q00 * (DC2 - DC8);
    if (num >= 0) {
        pred = (int) (((Q10<<7) + num) / (Q10<<8));
        if (Al > 0 && pred >= (1<<Al))
            pred = (1<<Al)-1;
    } else {
        pred = (int) (((Q10<<7) - num) / (Q10<<8));
        if (Al > 0 && pred >= (1<<Al))
            pred = (1<<Al)-1;
        pred = -pred;
    }
    workspace[8] = (JCOEF) pred;
}
/* AC20 */
if ((Al=coef_bits[3]) != 0 && workspace[16] == 0) {
    num = 9 * Q00 * (DC2 + DC8 - 2*DC5);
    if (num >= 0) {
        pred = (int) (((Q20<<7) + num) / (Q20<<8));
        if (Al > 0 && pred >= (1<<Al))
            pred = (1<<Al)-1;
    } else {
        pred = (int) (((Q20<<7) - num) / (Q20<<8));
        if (Al > 0 && pred >= (1<<Al))
            pred = (1<<Al)-1;
        pred = -pred;
    }
    workspace[16] = (JCOEF) pred;
}
/* AC11 */
if ((Al=coef_bits[4]) != 0 && workspace[9] == 0) {
    num = 5 * Q00 * (DC1 - DC3 - DC7 + DC9);
    if (num >= 0) {
        pred = (int) (((Q11<<7) + num) / (Q11<<8));
        if (Al > 0 && pred >= (1<<Al))
            pred = (1<<Al)-1;
    } else {
        pred = (int) (((Q11<<7) - num) / (Q11<<8));
        if (Al > 0 && pred >= (1<<Al))
            pred = (1<<Al)-1;
        pred = -pred;
    }
    workspace[9] = (JCOEF) pred;
}
/* AC02 */
if ((Al=coef_bits[5]) != 0 && workspace[2] == 0) {
    num = 9 * Q00 * (DC4 + DC6 - 2*DC5);
    if (num >= 0) {
        pred = (int) (((Q02<<7) + num) / (Q02<<8));
        if (Al > 0 && pred >= (1<<Al))
            pred = (1<<Al)-1;
    }
}

```

```

    } else {
        pred = (int) (((Q02<<7) - num) / (Q02<<8));
        if (A1 > 0 && pred >= (1<<A1))
            pred = (1<<A1)-1;
        pred = -pred;
    }
    workspace[2] = (JCOEF) pred;
}
/* OK, do the IDCT */
(*inverse_DCT) (cinfo, compptr, (JCOEFPTR) workspace,
                output_ptr, output_col);
/* Advance for next column */
DC1 = DC2; DC2 = DC3;
DC4 = DC5; DC5 = DC6;
DC7 = DC8; DC8 = DC9;
buffer_ptr++, prev_block_row++, next_block_row++;
output_col += compptr->DCT_scaled_size;
}
output_ptr += compptr->DCT_scaled_size;
}
}

if (++(cinfo->output_iMCU_row) < cinfo->total_iMCU_rows)
    return JPEG_ROW_COMPLETED;
return JPEG_SCAN_COMPLETED;
}

#endif /* BLOCK_SMOOTHING_SUPPORTED */

/*
 * Initialize coefficient buffer controller.
 */
GLOBAL(void)
jinit_d_coef_controller (j_decompress_ptr cinfo, boolean need_full_buffer)
{
    my_coef_ptr coef;

    coef = (my_coef_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                                   sizeof(my_coef_controller));
    cinfo->coef = (struct jpeg_d_coef_controller *) coef;
    coef->pub.start_input_pass = start_input_pass;
    coef->pub.start_output_pass = start_output_pass;
#ifdef BLOCK_SMOOTHING_SUPPORTED
    coef->coef_bits_latch = NULL;
#endif

    /* Create the coefficient buffer. */
    if (need_full_buffer) {
#ifdef D_MULTISCAN_FILES_SUPPORTED
        /* Allocate a full-image virtual array for each component, */
        /* padded to a multiple of samp_factor DCT blocks in each direction. */
        /* Note we ask for a pre-zeroed array. */
        int ci, access_rows;
        jpeg_component_info *compptr;

        for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
             ci++, compptr++) {
            access_rows = compptr->v_samp_factor;
#ifdef BLOCK_SMOOTHING_SUPPORTED
            /* If block smoothing could be used, need a bigger window */
            if (cinfo->progressive_mode)
                access_rows *= 3;
#endif
            coef->whole_image[ci] = (*cinfo->mem->request_virt_barray)
                ((j_common_ptr) cinfo, JPOOL_IMAGE, TRUE,
                 (JDIMENSION) jround_up((long) compptr->width_in_blocks,
                                         (long) compptr->h_samp_factor),
                 (JDIMENSION) jround_up((long) compptr->height_in_blocks,
                                         (long) compptr->v_samp_factor),
                 (JDIMENSION) access_rows);
        }
        coef->pub.consume_data = consume_data;
        coef->pub.decompress_data = decompress_data;
        coef->pub.coef_arrays = coef->whole_image; /* link to virtual arrays */
    }
    #else
        ERREXIT(cinfo, JERR_NOT_COMPILED);
    #endif
}

```

```

} else {
    /* We only need a single-MCU buffer. */
    JBLOCKROW buffer;
    int i;

    buffer = (JBLOCKROW)
        (*cinfo->mem->alloc_large) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            D_MAX_BLOCKS_IN_MCU * SIZEOF(JBLOCK));
    for (i = 0; i < D_MAX_BLOCKS_IN_MCU; i++) {
        coef->MCU_buffer[i] = buffer + i;
    }
    coef->pub.consume_data = dummy_consume_data;
    coef->pub.decompress_data = decompress_onepass;
    coef->pub.coef_arrays = NULL; /* flag for no virtual arrays */
}
}

```

```

/*
 * jdcolor.c
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains output colorspace conversion routines.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/* Private subobject */

typedef struct {
  struct jpeg_color_deconverter pub; /* public fields */

  /* Private state for YCC->RGB conversion */
  int * Cr_r_tab; /* => table for Cr to R conversion */
  int * Cb_b_tab; /* => table for Cb to B conversion */
  INT32 * Cr_g_tab; /* => table for Cr to G conversion */
  INT32 * Cb_g_tab; /* => table for Cb to G conversion */
} my_color_deconverter;

typedef my_color_deconverter * my_cconvert_ptr;

/***** YCbCr -> RGB conversion: most common case *****/
/*
 * YCbCr is defined per CCIR 601-1, except that Cb and Cr are
 * normalized to the range 0..MAXJSAMPLE rather than -0.5 .. 0.5.
 * The conversion equations to be implemented are therefore
 *   R = Y + 1.40200 * Cr
 *   G = Y - 0.34414 * Cb - 0.71414 * Cr
 *   B = Y + 1.77200 * Cb
 * where Cb and Cr represent the incoming values less CENTERJSAMPLE.
 * (These numbers are derived from TIFF 6.0 section 21, dated 3-June-92.)
 *
 * To avoid floating-point arithmetic, we represent the fractional constants
 * as integers scaled up by 2^16 (about 4 digits precision); we have to divide
 * the products by 2^16, with appropriate rounding, to get the correct answer.
 * Notice that Y, being an integral input, does not contribute any fraction
 * so it need not participate in the rounding.
 *
 * For even more speed, we avoid doing any multiplications in the inner loop
 * by precalculating the constants times Cb and Cr for all possible values.
 * For 8-bit JSAMPLEs this is very reasonable (only 256 entries per table);
 * for 12-bit samples it is still acceptable. It's not very reasonable for
 * 16-bit samples, but if you want lossless storage you shouldn't be changing
 * colorspace anyway.
 * The Cr=>R and Cb=>B values can be rounded to integers in advance; the
 * values for the G calculation are left scaled up, since we must add them
 * together before rounding.
 */

#define SCALEBITS 16 /* speediest right-shift on some machines */
#define ONE_HALF ((INT32) 1 << (SCALEBITS-1))
#define FIX(x) ((INT32) ((x) * (1L<<SCALEBITS) + 0.5))

/*
 * Initialize tables for YCC->RGB colorspace conversion.
 */

LOCAL(void)
build_ycc_rgb_table (j_decompress_ptr cinfo)
{
  my_cconvert_ptr cconvert = (my_cconvert_ptr) cinfo->cconvert;
  int i;
  INT32 x;
  SHIFT_TEMPS

  cconvert->Cr_r_tab = (int *)
    (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                              (MAXJSAMPLE+1) * sizeof(int));
  cconvert->Cb_b_tab = (int *)

```

```

        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
        (MAXJSAMPLE+1) * sizeof(int));
cconvert->Cr_g_tab = (INT32 *)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
        (MAXJSAMPLE+1) * sizeof(INT32));
cconvert->Cb_g_tab = (INT32 *)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
        (MAXJSAMPLE+1) * sizeof(INT32));

for (i = 0, x = -CENTERJSAMPLE; i <= MAXJSAMPLE; i++, x++) {
    /* i is the actual input pixel value, in the range 0..MAXJSAMPLE */
    /* The Cb or Cr value we are thinking of is x = i - CENTERJSAMPLE */
    /* Cr=>R value is nearest int to 1.40200 * x */
    cconvert->Cr_r_tab[i] = (int)
        RIGHT_SHIFT(FIX(1.40200) * x + ONE_HALF, SCALEBITS);
    /* Cb=>B value is nearest int to 1.77200 * x */
    cconvert->Cb_b_tab[i] = (int)
        RIGHT_SHIFT(FIX(1.77200) * x + ONE_HALF, SCALEBITS);
    /* Cr=>G value is scaled-up -0.71414 * x */
    cconvert->Cr_g_tab[i] = (- FIX(0.71414)) * x;
    /* Cb=>G value is scaled-up -0.34414 * x */
    /* We also add in ONE_HALF so that need not do it in inner loop */
    cconvert->Cb_g_tab[i] = (- FIX(0.34414)) * x + ONE_HALF;
}

/*
 * Convert some rows of samples to the output colorspace.
 *
 * Note that we change from noninterleaved, one-plane-per-component format
 * to interleaved-pixel format. The output buffer is therefore three times
 * as wide as the input buffer.
 * A starting row offset is provided only for the input buffer. The caller
 * can easily adjust the passed output_buf value to accommodate any row
 * offset required on that side.
 */

METHODDEF(void)
ycc_rgb_convert (j_decompress_ptr cinfo,
                 JSAMPIMAGE input_buf, JDIMENSION input_row,
                 JSAMPARRAY output_buf, int num_rows)
{
    my_cconvert_ptr cconvert = (my_cconvert_ptr) cinfo->cconvert;
    register int y, cb, cr;
    register JSAMPROW outptr;
    register JSAMPROW inptr0, inptr1, inptr2;
    register JDIMENSION col;
    JDIMENSION num_cols = cinfo->output_width;
    /* copy these pointers into registers if possible */
    register JSAMPLE * range_limit = cinfo->sample_range_limit;
    register int * Cr_rtab = cconvert->Cr_r_tab;
    register int * Cb_btab = cconvert->Cb_b_tab;
    register INT32 * Crgtab = cconvert->Cr_g_tab;
    register INT32 * Cbgtab = cconvert->Cb_g_tab;
    SHIFT_TEMPS

    while (--num_rows >= 0) {
        inptr0 = input_buf[0][input_row];
        inptr1 = input_buf[1][input_row];
        inptr2 = input_buf[2][input_row];
        input_row++;
        outptr = *output_buf++;
        for (col = 0; col < num_cols; col++) {
            y = GETJSAMPLE(inptr0[col]);
            cb = GETJSAMPLE(inptr1[col]);
            cr = GETJSAMPLE(inptr2[col]);
            /* Range-limiting is essential due to noise introduced by DCT losses. */
            outptr[RGB_RED] = range_limit[y + Cr_rtab[cr]];
            outptr[RGB_GREEN] = range_limit[y +
                ((int) RIGHT_SHIFT(Cbgtab[cb] + Crgtab[cr],
                SCALEBITS))];
            outptr[RGB_BLUE] = range_limit[y + Cb_btab[cb]];
            outptr += RGB_PIXELSIZE;
        }
    }
}

/***** Cases other than YCbCr -> RGB *****/

```

```

/*
 * Color conversion for no colorspace change: just copy the data,
 * converting from separate-planes to interleaved representation.
 */

```

```

METHODDEF(void)
null_convert (j_decompress_ptr cinfo,
              JSAMPIMAGE input_buf, JDIMENSION input_row,
              JSAMPARRAY output_buf, int num_rows)
{
    register JSAMPROW inptr, outptr;
    register JDIMENSION count;
    register int num_components = cinfo->num_components;
    JDIMENSION num_cols = cinfo->output_width;
    int ci;

    while (--num_rows >= 0) {
        for (ci = 0; ci < num_components; ci++) {
            inptr = input_buf[ci][input_row];
            outptr = output_buf[0] + ci;
            for (count = num_cols; count > 0; count--) {
                *outptr = *inptr++; /* needn't bother with GETJSAMPLE() here */
                outptr += num_components;
            }
            input_row++;
            output_buf++;
        }
    }
}

```

```

/*
 * Color conversion for grayscale: just copy the data.
 * This also works for YCbCr -> grayscale conversion, in which
 * we just copy the Y (luminance) component and ignore chrominance.
 */

```

```

METHODDEF(void)
grayscale_convert (j_decompress_ptr cinfo,
                  JSAMPIMAGE input_buf, JDIMENSION input_row,
                  JSAMPARRAY output_buf, int num_rows)
{
    jcopy_sample_rows(input_buf[0], (int) input_row, output_buf, 0,
                      num_rows, cinfo->output_width);
}

```

```

/*
 * Convert grayscale to RGB: just duplicate the graylevel three times.
 * This is provided to support applications that don't want to cope
 * with grayscale as a separate case.
 */

```

```

METHODDEF(void)
gray_rgb_convert (j_decompress_ptr cinfo,
                  JSAMPIMAGE input_buf, JDIMENSION input_row,
                  JSAMPARRAY output_buf, int num_rows)
{
    register JSAMPROW inptr, outptr;
    register JDIMENSION col;
    JDIMENSION num_cols = cinfo->output_width;

    while (--num_rows >= 0) {
        inptr = input_buf[0][input_row++];
        outptr = *output_buf++;
        for (col = 0; col < num_cols; col++) {
            /* We can dispense with GETJSAMPLE() here */
            outptr[RGB_RED] = outptr[RGB_GREEN] = outptr[RGB_BLUE] = inptr[col];
            outptr += RGB_PIXELSIZE;
        }
    }
}

```

```

/*
 * Adobe-style YCK->CMYK conversion.
 * We convert YCbCr to R=1-C, G=1-M, and B=1-Y using the same
 * conversion as above, while passing K (black) unchanged.
 */

```

```

* We assume build_ycc_rgb_table has been called.
*/

```

```

METHODDEF(void)

```

```

yccck_cmyk_convert (j_decompress_ptr cinfo,
                    JSAMPIMAGE input_buf, JDIMENSION input_row,
                    JSAMPARRAY output_buf, int num_rows)

```

```

{
    my_cconvert_ptr cconvert = (my_cconvert_ptr) cinfo->cconvert;
    register int y, cb, cr;
    register JSAMPROW outptr;
    register JSAMPROW inptr0, inptr1, inptr2, inptr3;
    register JDIMENSION col;
    JDIMENSION num_cols = cinfo->output_width;
    /* copy these pointers into registers if possible */
    register JSAMPLE * range_limit = cinfo->sample_range_limit;
    register int * Crrtab = cconvert->Cr_r_tab;
    register int * Cbbtab = cconvert->Cb_b_tab;
    register INT32 * Crgtab = cconvert->Cr_g_tab;
    register INT32 * Cbgtab = cconvert->Cb_g_tab;
    SHIFT_TEMPS

    while (--num_rows >= 0) {
        inptr0 = input_buf[0][input_row];
        inptr1 = input_buf[1][input_row];
        inptr2 = input_buf[2][input_row];
        inptr3 = input_buf[3][input_row];
        input_row++;
        outptr = *output_buf++;
        for (col = 0; col < num_cols; col++) {
            y = GETJSAMPLE(inptr0[col]);
            cb = GETJSAMPLE(inptr1[col]);
            cr = GETJSAMPLE(inptr2[col]);
            /* Range-limiting is essential due to noise introduced by DCT losses. */
            outptr[0] = range_limit[MAXJSAMPLE - (y + Crrtab[cr])]; /* red */
            outptr[1] = range_limit[MAXJSAMPLE - (y + Crgtab[cr] + Cbgtab[cb] + Cbbtab[cb])]; /* green */
            outptr[2] = range_limit[MAXJSAMPLE - (y + Cbbtab[cb])]; /* blue */
            /* K passes through unchanged */
            outptr[3] = inptr3[col]; /* don't need GETJSAMPLE here */
            outptr += 4;
        }
    }
}

```

```

/* Empty method for start_pass.
*/

```

```

METHODDEF(void)

```

```

start_pass_dcolor (j_decompress_ptr cinfo)

```

```

{
    /* no work needed */
}

```

```

/*
* Module initialization routine for output colorspace conversion.
*/

```

```

GLOBAL(void)

```

```

jinit_color_deconverter (j_decompress_ptr cinfo)

```

```

{
    my_cconvert_ptr cconvert;
    int ci;

    cconvert = (my_cconvert_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                                   sizeof(my_color_deconverter));
    cinfo->cconvert = (struct jpeg_color_deconverter *) cconvert;
    cconvert->pub.start_pass = start_pass_dcolor;

    /* Make sure num_components agrees with jpeg_color_space */
    switch (cinfo->jpeg_color_space) {
    case JCS_GRAYSCALE:
        if (cinfo->num_components != 1)
            ERREXIT(cinfo, JERR_BAD_J_COLORSPACE);
        break;
    }
}

```

```

case JCS_RGB:
case JCS_YCbCr:
    if (cinfo->num_components != 3)
        ERREXIT(cinfo, JERR_BAD_J_COLORSPACE);
    break;

case JCS_CMYK:
case JCS_YCCK:
    if (cinfo->num_components != 4)
        ERREXIT(cinfo, JERR_BAD_J_COLORSPACE);
    break;

default:
    /* JCS_UNKNOWN can be anything */
    if (cinfo->num_components < 1)
        ERREXIT(cinfo, JERR_BAD_J_COLORSPACE);
    break;
}

/* Set out_color_components and conversion method based on requested space.
 * Also clear the component_needed flags for any unused components,
 * so that earlier pipeline stages can avoid useless computation.
 */

switch (cinfo->out_color_space) {
case JCS_GRAYSCALE:
    cinfo->out_color_components = 1;
    if (cinfo->jpeg_color_space == JCS_GRAYSCALE ||
        cinfo->jpeg_color_space == JCS_YCbCr) {
        cconvert->pub.color_convert = grayscale_convert;
        /* For color->grayscale conversion, only the Y (0) component is needed */
        for (ci = 1; ci < cinfo->num_components; ci++)
            cinfo->comp_info[ci].component_needed = FALSE;
    } else
        ERREXIT(cinfo, JERR_CONVERSION_NOTIMPL);
    break;

case JCS_RGB:
    cinfo->out_color_components = RGB_PIXELSIZE;
    if (cinfo->jpeg_color_space == JCS_YCbCr) {
        cconvert->pub.color_convert = ycc_rgb_convert;
        build_ycc_rgb_table(cinfo);
    } else if (cinfo->jpeg_color_space == JCS_GRAYSCALE) {
        cconvert->pub.color_convert = gray_rgb_convert;
    } else if (cinfo->jpeg_color_space == JCS_RGB && RGB_PIXELSIZE == 3) {
        cconvert->pub.color_convert = null_convert;
    } else
        ERREXIT(cinfo, JERR_CONVERSION_NOTIMPL);
    break;

case JCS_CMYK:
    cinfo->out_color_components = 4;
    if (cinfo->jpeg_color_space == JCS_YCCK) {
        cconvert->pub.color_convert = ycck_cmyk_convert;
        build_ycc_rgb_table(cinfo);
    } else if (cinfo->jpeg_color_space == JCS_CMYK) {
        cconvert->pub.color_convert = null_convert;
    } else
        ERREXIT(cinfo, JERR_CONVERSION_NOTIMPL);
    break;

default:
    /* Permit null conversion to same output space */
    if (cinfo->out_color_space == cinfo->jpeg_color_space) {
        cinfo->out_color_components = cinfo->num_components;
        cconvert->pub.color_convert = null_convert;
    } else
        /* unsupported non-null conversion */
        ERREXIT(cinfo, JERR_CONVERSION_NOTIMPL);
    break;
}

if (cinfo->quantize_colors)
    cinfo->output_components = 1; /* single colormapped output component */
else
    cinfo->output_components = cinfo->out_color_components;
}

```



```

/*
 * jddctmgr.c
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains the inverse-DCT management logic.
 * This code selects a particular IDCT implementation to be used,
 * and it performs related housekeeping chores.  No code in this file
 * is executed per IDCT step, only during output pass setup.
 *
 * Note that the IDCT routines are responsible for performing coefficient
 * dequantization as well as the IDCT proper.  This module sets up the
 * dequantization multiplier table needed by the IDCT routine.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jdct.h"          /* Private declarations for DCT subsystem */

/*
 * The decompressor input side (jdinput.c) saves away the appropriate
 * quantization table for each component at the start of the first scan
 * involving that component.  (This is necessary in order to correctly
 * decode files that reuse Q-table slots.)
 * When we are ready to make an output pass, the saved Q-table is converted
 * to a multiplier table that will actually be used by the IDCT routine.
 * The multiplier table contents are IDCT-method-dependent.  To support
 * application changes in IDCT method between scans, we can remake the
 * multiplier tables if necessary.
 * In buffered-image mode, the first output pass may occur before any data
 * has been seen for some components, and thus before their Q-tables have
 * been saved away.  To handle this case, multiplier tables are preset
 * to zeroes; the result of the IDCT will be a neutral gray level.
 */

/* Private subobject for this module */
typedef struct {
  struct jpeg_inverse_dct pub; /* public fields */
  /* This array contains the IDCT method code that each multiplier table
   * is currently set up for, or -1 if it's not yet set up.
   * The actual multiplier tables are pointed to by dct_table in the
   * per-component comp_info structures.
   */
  int cur_method[MAX_COMPONENTS];
  my_idct_controller;
} my_idct_controller;

typedef my_idct_controller * my_idct_ptr;

/* Allocated multiplier tables: big enough for any supported variant */

typedef union {
  ISLOW_MULT_TYPE islow_array[DCTSIZE2];
#ifdef DCT_IFAST_SUPPORTED
  IFAST_MULT_TYPE ifast_array[DCTSIZE2];
#endif
#ifdef DCT_FLOAT_SUPPORTED
  FLOAT_MULT_TYPE float_array[DCTSIZE2];
#endif
} multiplier_table;

/* The current scaled-IDCT routines require ISLOW-style multiplier tables,
 * so be sure to compile that code if either ISLOW or SCALING is requested.
 */
#ifdef DCT_ISLOW_SUPPORTED
#define PROVIDE_ISLOW_TABLES
#else
#ifdef IDCT_SCALING_SUPPORTED
#define PROVIDE_ISLOW_TABLES
#endif
#endif

```

```

/*
 * Prepare for an output pass.
 * Here we select the proper IDCT routine for each component and build
 * a matching multiplier table.
 */

```

```

METHODDEF(void)
start_pass (j_decompress_ptr cinfo)
{
    my_idct_ptr idct = (my_idct_ptr) cinfo->idct;
    int ci, i;
    jpeg_component_info *comp_ptr;
    int method = 0;
    inverse_DCT_method_ptr method_ptr = NULL;
    JQUANT_TBL * qtbl;

    for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
        ci++, comp_ptr++) {
        /* Select the proper IDCT routine for this component's scaling */
        switch (comp_ptr->DCT_scaled_size) {
#ifdef IDCT_SCALING_SUPPORTED
        case 1:
            method_ptr = jpeg_idct_1x1;
            method = JDCT_ISLOW; /* jidctred uses islow-style table */
            break;
        case 2:
            method_ptr = jpeg_idct_2x2;
            method = JDCT_ISLOW; /* jidctred uses islow-style table */
            break;
        case 4:
            method_ptr = jpeg_idct_4x4;
            method = JDCT_ISLOW; /* jidctred uses islow-style table */
            break;
#endif
        case DCTSIZE:
            switch (cinfo->dct_method) {
#ifdef DCT_ISLOW_SUPPORTED
            case JDCT_ISLOW:
                method_ptr = jpeg_idct_islow;
                method = JDCT_ISLOW;
                break;
#endif
#ifdef DCT_IFAST_SUPPORTED
            case JDCT_IFAST:
                method_ptr = jpeg_idct_ifast;
                method = JDCT_IFAST;
                break;
#endif
#ifdef DCT_FLOAT_SUPPORTED
            case JDCT_FLOAT:
                method_ptr = jpeg_idct_float;
                method = JDCT_FLOAT;
                break;
#endif
            default:
                ERREXIT(cinfo, JERR_NOT_COMPILED);
                break;
            }
            break;
        default:
            ERREXIT1(cinfo, JERR_BAD_DCTSIZE, comp_ptr->DCT_scaled_size);
            break;
        }
        idct->pub.inverse_DCT[ci] = method_ptr;
        /* Create multiplier table from quant table.
         * However, we can skip this if the component is uninteresting
         * or if we already built the table. Also, if no quant table
         * has yet been saved for the component, we leave the
         * multiplier table all-zero; we'll be reading zeroes from the
         * coefficient controller's buffer anyway.
         */
        if (! comp_ptr->component_needed || idct->cur_method[ci] == method)
            continue;
        qtbl = comp_ptr->quant_table;
        if (qtbl == NULL) /* happens if no data yet for component */
            continue;
        idct->cur_method[ci] = method;
        switch (method) {
#ifdef PROVIDE_ISLOW_TABLES

```

```

case JDCT_ISLOW:
{
/* For LL&M IDCT method, multipliers are equal to raw quantization
 * coefficients, but are stored as ints to ensure access efficiency.
 */
ISLOW_MULT_TYPE * ismtbl = (ISLOW_MULT_TYPE *) compptr->dct_table;
for (i = 0; i < DCTSIZE2; i++) {
    ismtbl[i] = (ISLOW_MULT_TYPE) qtbl->quantval[i];
}
}
break;
#endif
#ifdef DCT_IFAST_SUPPORTED
case JDCT_IFAST:
{
/* For AA&N IDCT method, multipliers are equal to quantization
 * coefficients scaled by scalefactor[row]*scalefactor[col], where
 *   scalefactor[0] = 1
 *   scalefactor[k] = cos(k*PI/16) * sqrt(2)    for k=1..7
 * For integer operation, the multiplier table is to be scaled by
 *   IFAST_SCALE_BITS.
 */
IFAST_MULT_TYPE * ifmtbl = (IFAST_MULT_TYPE *) compptr->dct_table;
#define CONST_BITS 14
static const INT16 aanscales[DCTSIZE2] = {
/* precomputed values scaled up by 14 bits */
16384, 22725, 21407, 19266, 16384, 12873, 8867, 4520,
22725, 31521, 29692, 26722, 22725, 17855, 12299, 6270,
21407, 29692, 27969, 25172, 21407, 16819, 11585, 5906,
19266, 26722, 25172, 22654, 19266, 15137, 10426, 5315,
16384, 22725, 21407, 19266, 16384, 12873, 8867, 4520,
12873, 17855, 16819, 15137, 12873, 10114, 6967, 3552,
8867, 12299, 11585, 10426, 8867, 6967, 4799, 2446,
4520, 6270, 5906, 5315, 4520, 3552, 2446, 1247
};
SHIFT_TEMPS
for (i = 0; i < DCTSIZE2; i++) {
    ifmtbl[i] = (IFAST_MULT_TYPE)
        DESCALE(MULTIPLY16V16((INT32) qtbl->quantval[i],
            (INT32) aanscales[i]),
            CONST_BITS-IFAST_SCALE_BITS);
}
}
break;
#endif
#ifdef DCT_FLOAT_SUPPORTED
case JDCT_FLOAT:
{
/* For float AA&N IDCT method, multipliers are equal to quantization
 * coefficients scaled by scalefactor[row]*scalefactor[col], where
 *   scalefactor[0] = 1
 *   scalefactor[k] = cos(k*PI/16) * sqrt(2)    for k=1..7
 */
FLOAT_MULT_TYPE * ffmtbl = (FLOAT_MULT_TYPE *) compptr->dct_table;
int row, col;
static const double aanscalefactor[DCTSIZE] = {
1.0, 1.387039845, 1.306562965, 1.175875602,
1.0, 0.785694958, 0.541196100, 0.275899379
};
i = 0;
for (row = 0; row < DCTSIZE; row++) {
    for (col = 0; col < DCTSIZE; col++) {
        ffmtbl[i] = (FLOAT_MULT_TYPE)
            ((double) qtbl->quantval[i] *
            aanscalefactor[row] * aanscalefactor[col]);
        i++;
    }
}
break;
#endif
default:
ERREXIT(cinfo, JERR_NOT_COMPILED);
break;
}
}
}

```

```

/*
 * Initialize IDCT manager.
 */

GLOBAL(void)
jinit_inverse_dct (j_decompress_ptr cinfo)
{
    my_idct_ptr idct;
    int ci;
    jpeg_component_info *comp_ptr;

    idct = (my_idct_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                                   sizeof(my_idct_controller));
    cinfo->idct = (struct jpeg_inverse_dct *) idct;
    idct->pub.start_pass = start_pass;

    for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
         ci++, comp_ptr++) {
        /* Allocate and pre-zero a multiplier table for each component */
        comp_ptr->dct_table =
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                                       sizeof(multiplier_table));
        MEMZERO(comp_ptr->dct_table, sizeof(multiplier_table));
        /* Mark multiplier table not yet set up for any method */
        idct->cur_method[ci] = -1;
    }
}

```

```

/*
 * jdhuuff.c
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains Huffman entropy decoding routines.
 *
 * Much of the complexity here has to do with supporting input suspension.
 * If the data source module demands suspension, we want to be able to back
 * up to the start of the current MCU. To do this, we copy state variables
 * into local working storage, and update them back to the permanent
 * storage only upon successful completion of an MCU.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jdhuuff.h" /* Declarations shared with jdphuff.c */

/*
 * Expanded entropy decoder object for Huffman decoding.
 *
 * The savable_state subrecord contains fields that change within an MCU,
 * but must not be updated permanently until we complete the MCU.
 */

typedef struct {
  int last_dc_val[MAX_COMPS_IN_SCAN]; /* last DC coef for each component */
} savable_state;

/* This macro is to work around compilers with missing or broken
 * structure assignment. You'll need to fix this code if you have
 * such a compiler and you change MAX_COMPS_IN_SCAN.
 */
#ifdef NO_STRUCT_ASSIGN
#define ASSIGN_STATE(dest,src) ((dest) = (src))
#else
#define ASSIGN_STATE(dest,src) \
  if (MAX_COMPS_IN_SCAN == 4) \
  define ASSIGN_STATE(dest,src) \
  = ((dest).last_dc_val[0] = (src).last_dc_val[0], \
    (dest).last_dc_val[1] = (src).last_dc_val[1], \
    (dest).last_dc_val[2] = (src).last_dc_val[2], \
    (dest).last_dc_val[3] = (src).last_dc_val[3]) \
  #endif
#endif

typedef struct {
  struct jpeg_entropy_decoder pub; /* public fields */

  /* These fields are loaded into local variables at start of each MCU.
   * In case of suspension, we exit WITHOUT updating them.
   */
  bitread_perm_state bitstate; /* Bit buffer at start of MCU */
  savable_state saved; /* Other state at start of MCU */

  /* These fields are NOT loaded into local working state. */
  unsigned int restarts_to_go; /* MCUs left in this restart interval */

  /* Pointers to derived tables (these workspaces have image lifespan) */
  d_derived_tbl * dc_derived_tbls[NUM_HUFF_TBLS];
  d_derived_tbl * ac_derived_tbls[NUM_HUFF_TBLS];

  /* Precalculated info set up by start_pass for use in decode_mcu: */

  /* Pointers to derived tables to be used for each block within an MCU */
  d_derived_tbl * dc_cur_tbls[D_MAX_BLOCKS_IN_MCU];
  d_derived_tbl * ac_cur_tbls[D_MAX_BLOCKS_IN_MCU];
  /* Whether we care about the DC and AC coefficient values for each block */
  boolean dc_needed[D_MAX_BLOCKS_IN_MCU];
  boolean ac_needed[D_MAX_BLOCKS_IN_MCU];
} huff_entropy_decoder;

typedef huff_entropy_decoder * huff_entropy_ptr;

```

```

/*
 * Initialize for a Huffman-compressed scan.
 */

METHODDEF(void)
start_pass_huff_decoder (j_decompress_ptr cinfo)
{
    huff_entropy_ptr entropy = (huff_entropy_ptr) cinfo->entropy;
    int ci, blkcn, dctbl, actbl;
    jpeg_component_info * compptr;

    /* Check that the scan parameters Ss, Se, Ah/Al are OK for sequential JPEG.
     * This ought to be an error condition, but we make it a warning because
     * there are some baseline files out there with all zeroes in these bytes.
     */
    if (cinfo->Ss != 0 || cinfo->Se != DCTSIZE2-1 ||
        cinfo->Ah != 0 || cinfo->Al != 0)
        WARNMS(cinfo, JWRN_NOT_SEQUENTIAL);

    for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
        compptr = cinfo->cur_comp_info[ci];
        dctbl = compptr->dc_tbl_no;
        actbl = compptr->ac_tbl_no;
        /* Compute derived values for Huffman tables */
        /* We may do this more than once for a table, but it's not expensive */
        jpeg_make_d_derived_tbl(cinfo, TRUE, dctbl,
                                & entropy->dc_derived_tbls[dctbl]);
        jpeg_make_d_derived_tbl(cinfo, FALSE, actbl,
                                & entropy->ac_derived_tbls[actbl]);
        /* Initialize DC predictions to 0 */
        entropy->saved.last_dc_val[ci] = 0;

        /* Precalculate decoding info for each block in an MCU of this scan */
        for (blkcn = 0; blkcn < cinfo->blocks_in_MCU; blkcn++) {
            ci = cinfo->MCU_membership[blkcn];
            compptr = cinfo->cur_comp_info[ci];
            /* Precalculate which table to use for each block */
            entropy->dc_cur_tbls[blkcn] = entropy->dc_derived_tbls[compptr->dc_tbl_no];
            entropy->ac_cur_tbls[blkcn] = entropy->ac_derived_tbls[compptr->ac_tbl_no];
            /* Decide whether we really care about the coefficient values */
            if (compptr->component_needed) {
                entropy->dc_needed[blkcn] = TRUE;
                /* we don't need the ACs if producing a 1/8th-size image */
                entropy->ac_needed[blkcn] = (compptr->DCT_scaled_size > 1);
            } else {
                entropy->dc_needed[blkcn] = entropy->ac_needed[blkcn] = FALSE;
            }
        }

        /* Initialize bitread state variables */
        entropy->bitstate.bits_left = 0;
        entropy->bitstate.get_buffer = 0; /* unnecessary, but keeps Purify quiet */
        entropy->pub.insufficient_data = FALSE;

        /* Initialize restart counter */
        entropy->restarts_to_go = cinfo->restart_interval;
    }

    /*
     * Compute the derived values for a Huffman table.
     * This routine also performs some validation checks on the table.
     * Note this is also used by jdphuff.c.
     */

    GLOBAL(void)
    jpeg_make_d_derived_tbl (j_decompress_ptr cinfo, boolean isDC, int tblno,
                             d_derived_tbl ** pdtbl)
    {
        JHUFF_TBL *htbl;
        d_derived_tbl *dtbl;
        int p, i, l, si, numsymbols;
        int lookbits, ctr;
        char huffsize[257];
        unsigned int huffcode[257];
        unsigned int code;

        /* Note that huffsize[] and huffcode[] are filled in code-length order,

```

```

    * paralleling the order of the symbols themselves in htbl->huffval[].
    */

/* Find the input Huffman table */
if (tblno < 0 || tblno >= NUM_HUFF_TBLS)
    ERREXIT1(cinfo, JERR_NO_HUFF_TABLE, tblno);
htbl =
    isDC ? cinfo->dc_huff_tbl_ptrs[tblno] : cinfo->ac_huff_tbl_ptrs[tblno];
if (htbl == NULL)
    ERREXIT1(cinfo, JERR_NO_HUFF_TABLE, tblno);

/* Allocate a workspace if we haven't already done so. */
if (*pdtbl == NULL)
    *pdtbl = (d_derived_tbl *)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            sizeof(d_derived_tbl));
dtbl = *pdtbl;
dtbl->pub = htbl; /* fill in back link */

/* Figure C.1: make table of Huffman code length for each symbol */
p = 0;
for (l = 1; l <= 16; l++) {
    i = (int) htbl->bits[l];
    if (i < 0 || p + i > 256) /* protect against table overrun */
        ERREXIT(cinfo, JERR_BAD_HUFF_TABLE);
    while (i--)
        huffsize[p++] = (char) l;
}
huffsize[p] = 0;
numsymbols = p;

/* Figure C.2: generate the codes themselves */
/* We also validate that the counts represent a legal Huffman code tree. */
code = 0;
si = huffsize[0];
p = 0;
while (huffsize[p]) {
    while (((int) huffsize[p]) == si) {
        huffcode[p++] = code;
        code++;
    }
    /* code is now 1 more than the last code used for codelength si; but
     * it must still fit in si bits, since no code is allowed to be all ones.
     */
    if (((INT32) code) >= (((INT32) 1) << si))
        ERREXIT(cinfo, JERR_BAD_HUFF_TABLE);
    code <<= 1;
    si++;
}

/* Figure F.15: generate decoding tables for bit-sequential decoding */
p = 0;
for (l = 1; l <= 16; l++) {
    if (htbl->bits[l]) {
        /* valoffset[l] = huffval[] index of 1st symbol of code length l,
         * minus the minimum code of length l
         */
        dtbl->valoffset[l] = (INT32) p - (INT32) huffcode[p];
        p += htbl->bits[l];
        dtbl->maxcode[l] = huffcode[p-1]; /* maximum code of length l */
    } else {
        dtbl->maxcode[l] = -1; /* -1 if no codes of this length */
    }
}
dtbl->maxcode[17] = 0xFFFFFL; /* ensures jpeg_huff_decode terminates */

/* Compute lookahead tables to speed up decoding.
 * First we set all the table entries to 0, indicating "too long";
 * then we iterate through the Huffman codes that are short enough and
 * fill in all the entries that correspond to bit sequences starting
 * with that code.
 */
MEMZERO(dtbl->look_nbits, sizeof(dtbl->look_nbits));
p = 0;
for (l = 1; l <= HUFF_LOOKAHEAD; l++) {

```

```

    for (i = 1; i <= (int) htbl->bits[i]; i++, p++) {
        /* l = current code's length, p = its index in huffcode[] & huffval[] */
        /* Generate left-justified code followed by all possible bit sequences */
        lookbits = huffcode[p] << (HUFF_LOOKAHEAD-1);
        for (ctr = 1 << (HUFF_LOOKAHEAD-1); ctr > 0; ctr--) {
            dtbl->look_nbits[lookbits] = 1;
            dtbl->look_sym[lookbits] = htbl->huffval[p];
            lookbits++;
        }
    }
}

/* Validate symbols as being reasonable.
 * For AC tables, we make no check, but accept all byte values 0..255.
 * For DC tables, we require the symbols to be in range 0..15.
 * (Tighter bounds could be applied depending on the data depth and mode,
 * but this is sufficient to ensure safe decoding.)
 */
if (isDC) {
    for (i = 0; i < numsymbols; i++) {
        int sym = htbl->huffval[i];
        if (sym < 0 || sym > 15)
            ERREXIT(cinfo, JERR_BAD_HUFF_TABLE);
    }
}

/*
 * Out-of-line code for bit fetching (shared with jdphuff.c).
 * See jdphuff.h for info about usage.
 * Note: current values of get_buffer and bits_left are passed as parameters,
 * but are returned in the corresponding fields of the state struct.
 *
 * On most machines MIN_GET_BITS should be 25 to allow the full 32-bit width
 * of get_buffer to be used. (On machines with wider words, an even larger
 * buffer could be used.) However, on some machines 32-bit shifts are
 * quite slow and take time proportional to the number of places shifted.
 * (This is true with most PC compilers, for instance.) In this case it may
 * be a win to set MIN_GET_BITS to the minimum value of 15. This reduces the
 * average shift distance at the cost of more calls to jpeg_fill_bit_buffer.
 */

#ifdef SLOW_SHIFT_32
#define MIN_GET_BITS 15 /* minimum allowable value */
#else
#define MIN_GET_BITS (BIT_BUF_SIZE-7)
#endif

GLOBAL(boolean)
jpeg_fill_bit_buffer (bitread_working_state * state,
                     register bit_buf_type get_buffer, register int bits_left,
                     int nbits)
/* Load up the bit buffer to a depth of at least nbits */
{
    /* Copy heavily used state fields into locals (hopefully registers) */
    register const JOCTET * next_input_byte = state->next_input_byte;
    register size_t bytes_in_buffer = state->bytes_in_buffer;
    jpeg_decompress_ptr cinfo = state->cinfo;

    /* Attempt to load at least MIN_GET_BITS bits into get_buffer. */
    /* (It is assumed that no request will be for more than that many bits.) */
    /* We fail to do so only if we hit a marker or are forced to suspend. */

    if (cinfo->unread_marker == 0) { /* cannot advance past a marker */
        while (bits_left < MIN_GET_BITS) {
            register int c;

            /* Attempt to read a byte */
            if (bytes_in_buffer == 0) {
                if (! (*cinfo->src->fill_input_buffer) (cinfo))
                    return FALSE;
                next_input_byte = cinfo->src->next_input_byte;
                bytes_in_buffer = cinfo->src->bytes_in_buffer;
            }
            bytes_in_buffer--;
            c = GETJOCTET(*next_input_byte++);

            /* If it's 0xFF, check and discard stuffed zero byte */

```



```

    if (c == 0xFF) {
/* Loop here to discard any padding FF's on terminating marker,
 * so that we can save a valid unread_marker value. NOTE: we will
 * accept multiple FF's followed by a 0 as meaning a single FF data
 * byte. This data pattern is not valid according to the standard.
 */
do {
    if (bytes_in_buffer == 0) {
        if (!(*cinfo->src->fill_input_buffer) (cinfo))
            return FALSE;
        next_input_byte = cinfo->src->next_input_byte;
        bytes_in_buffer = cinfo->src->bytes_in_buffer;
    }
    bytes_in_buffer--;
    c = GETJOCTET(*next_input_byte++);
} while (c == 0xFF);

if (c == 0) {
/* Found FF/00, which represents an FF data byte */
    c = 0xFF;
} else {
/* Oops, it's actually a marker indicating end of compressed data.
 * Save the marker code for later use.
 * Fine point: it might appear that we should save the marker into
 * bitread working state, not straight into permanent state. But
 * once we have hit a marker, we cannot need to suspend within the
 * current MCU, because we will read no more bytes from the data
 * source. So it is OK to update permanent state right away.
 */
    cinfo->unread_marker = c;
/* See if we need to insert some fake zero bits. */
    goto no_more_bytes;
}

/* OK, load c into get_buffer */
get_buffer = (get_buffer << 8) | c;
bits_left += 8;
} /* end while */
} else {
no_more_bytes:
/* We get here if we've read the marker that terminates the compressed
 * data segment. There should be enough bits in the buffer register
 * to satisfy the request; if so, no problem.
 */
if (nbits > bits_left) {
/* Uh-oh. Report corrupted data to user and stuff zeroes into
 * the data stream, so that we can produce some kind of image.
 * We use a nonvolatile flag to ensure that only one warning message
 * appears per data segment.
 */
    if (!cinfo->entropy->insufficient_data) {
        WARNMS(cinfo, JWRN_HIT_MARKER);
        cinfo->entropy->insufficient_data = TRUE;
    }
/* Fill the buffer with zero bits */
    get_buffer <=<= MIN_GET_BITS - bits_left;
    bits_left = MIN_GET_BITS;
}

/* Unload the local registers */
state->next_input_byte = next_input_byte;
state->bytes_in_buffer = bytes_in_buffer;
state->get_buffer = get_buffer;
state->bits_left = bits_left;

return TRUE;
}

/*
 * Out-of-line code for Huffman code decoding.
 * See jd Huff.h for info about usage.
 */

GLOBAL(int)
jpeg_huff_decode (bitread_working_state * state,
    register bit_buf_type get_buffer, register int bits_left,
    d_derived_tbl * htbl, int min_bits)

```

```

{
    register int l = min_bits;
    register INT32 code;

    /* HUFF_DECODE has determined that the code is at least min_bits */
    /* bits long, so fetch that many bits in one swoop. */

    CHECK_BIT_BUFFER(*state, l, return -1);
    code = GET_BITS(l);

    /* Collect the rest of the Huffman code one bit at a time. */
    /* This is per Figure F.16 in the JPEG spec. */

    while (code > htbl->maxcode[l]) {
        code <<= 1;
        CHECK_BIT_BUFFER(*state, 1, return -1);
        code |= GET_BITS(1);
        l++;
    }

    /* Unload the local registers */
    state->get_buffer = get_buffer;
    state->bits_left = bits_left;

    /* With garbage input we may reach the sentinel value l = 17. */
    if (l > 16) {
        WARNMS(state->cinfo, JWRN_HUFF_BAD_CODE);
        return 0; /* fake a zero as the safest result */
    }

    return htbl->pub->huffval[ (int) (code + htbl->valoffset[l]) ];
}

```

Figure F.12: extend sign bit.

On some machines, a shift and add will be faster than a table lookup.

```

#ifdef AVOID_TABLES
#define HUFF_EXTEND(x,s) ((x) < (1<<((s)-1)) ? (x) + (((-1)<<(s)) + 1) : (x))
#else
#define HUFF_EXTEND(x,s) ((x) < extend_test[s] ? (x) + extend_offset[s] : (x))

static const int extend_test[16] = /* entry n is 2**(n-1) */
{ 0, 0x0001, 0x0002, 0x0004, 0x0008, 0x0010, 0x0020, 0x0040, 0x0080,
  0x0100, 0x0200, 0x0400, 0x0800, 0x1000, 0x2000, 0x4000 };

static const int extend_offset[16] = /* entry n is (-1 << n) + 1 */
{ 0, ((-1)<<1) + 1, ((-1)<<2) + 1, ((-1)<<3) + 1, ((-1)<<4) + 1,
  ((-1)<<5) + 1, ((-1)<<6) + 1, ((-1)<<7) + 1, ((-1)<<8) + 1,
  ((-1)<<9) + 1, ((-1)<<10) + 1, ((-1)<<11) + 1, ((-1)<<12) + 1,
  ((-1)<<13) + 1, ((-1)<<14) + 1, ((-1)<<15) + 1 };

#endif /* AVOID_TABLES */

/*
 * Check for a restart marker & resynchronize decoder.
 * Returns FALSE if must suspend.
 */

LOCAL(boolean)
process_restart (j_decompress_ptr cinfo)
{
    huff_entropy_ptr entropy = (huff_entropy_ptr) cinfo->entropy;
    int ci;

    /* Throw away any unused bits remaining in bit buffer; */
    /* include any full bytes in next_marker's count of discarded bytes */
    cinfo->marker->discarded_bytes += entropy->bitstate.bits_left / 8;
    entropy->bitstate.bits_left = 0;

    /* Advance past the RSTn marker */
    if (!(*cinfo->marker->read_restart_marker)(cinfo))
        return FALSE;
}

```

```

/* Re-initialize DC predictions to 0 */
for (ci = 0; ci < cinfo->comps_in_scan; ci++)
    entropy->saved.last_dc_val[ci] = 0;

/* Reset restart counter */
entropy->restarts_to_go = cinfo->restart_interval;

/* Reset out-of-data flag, unless read_restart_marker left us smack up
 * against a marker. In that case we will end up treating the next data
 * segment as empty, and we can avoid producing bogus output pixels by
 * leaving the flag set.
 */
if (cinfo->unread_marker == 0)
    entropy->pub.insufficient_data = FALSE;

return TRUE;
}

/*
 * Decode and return one MCU's worth of Huffman-compressed coefficients.
 * The coefficients are reordered from zigzag order into natural array order,
 * but are not dequantized.
 *
 * The i'th block of the MCU is stored into the block pointed to by
 * MCU_data[i]. WE ASSUME THIS AREA HAS BEEN ZEROED BY THE CALLER.
 * (Wholesale zeroing is usually a little faster than retail...)
 *
 * Returns FALSE if data source requested suspension. In that case no
 * changes have been made to permanent state. (Exception: some output
 * coefficients may already have been assigned. This is harmless for
 * this module, since we'll just re-assign them on the next call.)
 */
METHODDEF(boolean)
decode_mcu (j_decompress_ptr cinfo, JBLOCKROW *MCU_data)
{
    huff_entropy_ptr entropy = (huff_entropy_ptr) cinfo->entropy;
    int blkn;
    BITREAD_STATE_VARS;
    savable_state state;

    /* Process restart marker if needed; may have to suspend */
    if (cinfo->restart_interval) {
        if (entropy->restarts_to_go == 0)
            if (! process_restart(cinfo))
                return FALSE;
    }

    /* If we've run out of data, just leave the MCU set to zeroes.
     * This way, we return uniform gray for the remainder of the segment.
     */
    if (! entropy->pub.insufficient_data) {
        /* Load up working state */
        BITREAD_LOAD_STATE(cinfo, entropy->bitstate);
        ASSIGN_STATE(state, entropy->saved);

        /* Outer loop handles each block in the MCU */

        for (blkn = 0; blkn < cinfo->blocks_in_MCU; blkn++) {
            JBLOCKROW block = MCU_data[blkn];
            d_derived_tbl * dctbl = entropy->dc_cur_tbls[blkn];
            d_derived_tbl * actbl = entropy->ac_cur_tbls[blkn];
            register int s, k, r;

            /* Decode a single block's worth of coefficients */

            /* Section F.2.2.1: decode the DC coefficient difference */
            HUFF_DECODE(s, br_state, dctbl, return FALSE, label1);
            if (s) {
                CHECK_BIT_BUFFER(br_state, s, return FALSE);
                r = GET_BITS(s);
                s = HUFF_EXTEND(r, s);
            }

            if (entropy->dc_needed[blkn]) {
                /* Convert DC difference to actual value, update last_dc_val */
                int ci = cinfo->MCU_membership[blkn];

```

```

s += state.last_dc_val[ci];
state.last_dc_val[ci] = s;
/* Output the DC coefficient (assumes jpeg_natural_order[0] = 0) */
(*block)[0] = (JCOEF) s;
}

if (entropy->ac_needed[blkn]) {

/* Section F.2.2.2: decode the AC coefficients */
/* Since zeroes are skipped, output area must be cleared beforehand */
for (k = 1; k < DCTSIZE2; k++) {
    HUFF_DECODE(s, br_state, actbl, return FALSE, label2);

    r = s >> 4;
    s &= 15;

    if (s) {
        k += r;
        CHECK_BIT_BUFFER(br_state, s, return FALSE);
        r = GET_BITS(s);
        s = HUFF_EXTEND(r, s);
        /* Output coefficient in natural (dezigzagged) order.
         * Note: the extra entries in jpeg_natural_order[] will save us
         * if k >= DCTSIZE2, which could happen if the data is corrupted.
         */
        (*block)[jpeg_natural_order[k]] = (JCOEF) s;
    } else {
        if (r != 15)
            break;
        k += 15;
    }
}

} else {

/* Section F.2.2.2: decode the AC coefficients */
/* In this path we just discard the values */
for (k = 1; k < DCTSIZE2; k++) {
    HUFF_DECODE(s, br_state, actbl, return FALSE, label3);

    r = s >> 4;
    s &= 15;

    if (s) {
        k += r;
        CHECK_BIT_BUFFER(br_state, s, return FALSE);
        DROP_BITS(s);
    } else {
        if (r != 15)
            break;
        k += 15;
    }
}

}

/* Completed MCU, so update state */
BITREAD_SAVE_STATE(cinfo, entropy->bitstate);
ASSIGN_STATE(entropy->saved, state);
}

/* Account for restart interval (no-op if not using restarts) */
entropy->restarts_to_go--;

return TRUE;
}

/*
 * Module initialization routine for Huffman entropy decoding.
 */

GLOBAL(void)
jinit_huff_decoder (j_decompress_ptr cinfo)
{
    huff_entropy_ptr entropy;
    int i;

    entropy = (huff_entropy_ptr)

```

```

    (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                                sizeof(huff_entropy_decoder));
    cinfo->entropy = (struct jpeg_entropy_decoder *) entropy;
    entropy->pub.start_pass = start_pass_huff_decoder;
    entropy->pub.decode_mcu = decode_mcu;

    /* Mark tables unallocated */
    for (i = 0; i < NUM_HUFF_TBLS; i++) {
        entropy->dc_derived_tbls[i] = entropy->ac_derived_tbls[i] = NULL;
    }
}

```

```

/*
 * jinput.c
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains input control logic for the JPEG decompressor.
 * These routines are concerned with controlling the decompressor's input
 * processing (marker reading and coefficient decoding). The actual input
 * reading is done in jdmarker.c, jdhuft.c, and jdphuff.c.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/* Private state */

typedef struct {
    struct jpeg_input_controller pub; /* public fields */

    boolean inheaders; /* TRUE until first SOS is reached */
} my_input_controller;

typedef my_input_controller * my_inputctl_ptr;

/* Forward declarations */
METHODDEF(int) consume_markers JPP((j_decompress_ptr cinfo));

/*
 * Routines to calculate various quantities related to the size of the image.
 */

LOCAL(void)
initial_setup (j_decompress_ptr cinfo)
/* Called once, when first SOS marker is reached */
{
    int ci;
    jpeg_component_info *comp_ptr;

    /* Make sure image isn't bigger than I can handle */
    if ((long) cinfo->image_height > (long) JPEG_MAX_DIMENSION ||
        (long) cinfo->image_width > (long) JPEG_MAX_DIMENSION)
        ERREXIT1(cinfo, JERR_IMAGE_TOO_BIG, (unsigned int) JPEG_MAX_DIMENSION);

    /* For now, precision must match compiled-in value... */
    if (cinfo->data_precision != BITS_IN_JSAMPLE)
        ERREXIT1(cinfo, JERR_BAD_PRECISION, cinfo->data_precision);

    /* Check that number of components won't exceed internal array sizes */
    if (cinfo->num_components > MAX_COMPONENTS)
        ERREXIT2(cinfo, JERR_COMPONENT_COUNT, cinfo->num_components,
            MAX_COMPONENTS);

    /* Compute maximum sampling factors; check factor validity */
    cinfo->max_h_samp_factor = 1;
    cinfo->max_v_samp_factor = 1;
    for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
        ci++, comp_ptr++) {
        if (comp_ptr->h_samp_factor <= 0 || comp_ptr->h_samp_factor > MAX_SAMP_FACTOR ||
            comp_ptr->v_samp_factor <= 0 || comp_ptr->v_samp_factor > MAX_SAMP_FACTOR)
            ERREXIT(cinfo, JERR_BAD_SAMPLING);
        cinfo->max_h_samp_factor = MAX(cinfo->max_h_samp_factor,
            comp_ptr->h_samp_factor);
        cinfo->max_v_samp_factor = MAX(cinfo->max_v_samp_factor,
            comp_ptr->v_samp_factor);
    }

    /* We initialize DCT_scaled_size and min_DCT_scaled_size to DCTSIZE.
     * In the full decompressor, this will be overridden by jdmaster.c;
     * but in the transcoder, jdmaster.c is not used, so we must do it here.
     */
    cinfo->min_DCT_scaled_size = DCTSIZE;

    /* Compute dimensions of components */
    for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;

```

```

    ci++, compptr++) {
    compptr->DCT_scaled_size = DCTSIZE;
    /* Size in DCT blocks */
    compptr->width_in_blocks = (JDIMENSION)
        jdiv_round_up((long) cinfo->image_width * (long) compptr->h_samp_factor,
            (long) (cinfo->max_h_samp_factor * DCTSIZE));
    compptr->height_in_blocks = (JDIMENSION)
        jdiv_round_up((long) cinfo->image_height * (long) compptr->v_samp_factor,
            (long) (cinfo->max_v_samp_factor * DCTSIZE));
    /* downscaled width and downscaled height will also be overridden by
     * jdmaster.c if we are doing full decompression. The transcoder library
     * doesn't use these values, but the calling application might.
     */
    /* Size in samples */
    compptr->downsampled_width = (JDIMENSION)
        jdiv_round_up((long) cinfo->image_width * (long) compptr->h_samp_factor,
            (long) cinfo->max_h_samp_factor);
    compptr->downsampled_height = (JDIMENSION)
        jdiv_round_up((long) cinfo->image_height * (long) compptr->v_samp_factor,
            (long) cinfo->max_v_samp_factor);
    /* Mark component needed, until color conversion says otherwise */
    compptr->component_needed = TRUE;
    /* Mark no quantization table yet saved for component */
    compptr->quant_table = NULL;
}

/* Compute number of fully interleaved MCU rows. */
cinfo->total_iMCU_rows = (JDIMENSION)
    jdiv_round_up((long) cinfo->image_height,
        (long) (cinfo->max_v_samp_factor * DCTSIZE));

/* Decide whether file contains multiple scans */
if (cinfo->comps_in_scan < cinfo->num_components || cinfo->progressive_mode)
    cinfo->inputctl->has_multiple_scans = TRUE;
else
    cinfo->inputctl->has_multiple_scans = FALSE;
}

LOCAL(void)
per_scan_setup (j_decompress_ptr cinfo)
/* Do computations that are needed before processing a JPEG scan */
/* cinfo->comps_in_scan and cinfo->cur_comp_info[] were set from SOS marker */
{
    int ci, mcublk, tmp;
    jpeg_component_info *compptr;

    if (cinfo->comps_in_scan == 1) {
        /* Noninterleaved (single-component) scan */
        compptr = cinfo->cur_comp_info[0];

        /* Overall image size in MCUs */
        cinfo->MCUs_per_row = compptr->width_in_blocks;
        cinfo->MCU_rows_in_scan = compptr->height_in_blocks;

        /* For noninterleaved scan, always one block per MCU */
        compptr->MCU_width = 1;
        compptr->MCU_height = 1;
        compptr->MCU_blocks = 1;
        compptr->MCU_sample_width = compptr->DCT_scaled_size;
        compptr->last_col_width = 1;
        /* For noninterleaved scans, it is convenient to define last_row_height
         * as the number of block rows present in the last iMCU row.
         */
        tmp = (int) (compptr->height_in_blocks % compptr->v_samp_factor);
        if (tmp == 0) tmp = compptr->v_samp_factor;
        compptr->last_row_height = tmp;

        /* Prepare array describing MCU composition */
        cinfo->blocks_in_MCU = 1;
        cinfo->MCU_membership[0] = 0;
    } else {
        /* Interleaved (multi-component) scan */
        if (cinfo->comps_in_scan <= 0 || cinfo->comps_in_scan > MAX_COMPS_IN_SCAN)
            ERREXIT2(cinfo, JERR_COMPONENT_COUNT, cinfo->comps_in_scan,
                MAX_COMPS_IN_SCAN);
    }
}

```

```

/* Overall image size in MCUs */
cinfo->MCUs_per_row = (JDIMENSION)
    jdiv_round_up((long) cinfo->image_width,
        (long) (cinfo->max_h_samp_factor*DCTSIZE));
cinfo->MCU_rows_in_scan = (JDIMENSION)
    jdiv_round_up((long) cinfo->image_height,
        (long) (cinfo->max_v_samp_factor*DCTSIZE));

cinfo->blocks_in_MCU = 0;

for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
    compptr = cinfo->cur_comp_info[ci];
    /* Sampling factors give # of blocks of component in each MCU */
    compptr->MCU_width = compptr->h_samp_factor;
    compptr->MCU_height = compptr->v_samp_factor;
    compptr->MCU_blocks = compptr->MCU_width * compptr->MCU_height;
    compptr->MCU_sample_width = compptr->MCU_width * compptr->DCT_scaled_size;
    /* Figure number of non-dummy blocks in last MCU column & row */
    tmp = (int) (compptr->width_in_blocks % compptr->MCU_width);
    if (tmp == 0) tmp = compptr->MCU_width;
    compptr->last_col_width = tmp;
    tmp = (int) (compptr->height_in_blocks % compptr->MCU_height);
    if (tmp == 0) tmp = compptr->MCU_height;
    compptr->last_row_height = tmp;
    /* Prepare array describing MCU composition */
    mcublks = compptr->MCU_blocks;
    if (cinfo->blocks_in_MCU + mcublks > D_MAX_BLOCKS_IN_MCU)
        ERREXIT(cinfo, JERR_BAD_MCU_SIZE);
    while (mcublks-- > 0) {
        cinfo->MCU_membership[cinfo->blocks_in_MCU++] = ci;
    }
}

```

```

/* Save away a copy of the Q-table referenced by each component present
in the current scan, unless already saved during a prior scan.

In a multiple-scan JPEG file, the encoder could assign different components
the same Q-table slot number, but change table definitions between scans
so that each component uses a different Q-table. (The IJG encoder is not
currently capable of doing this, but other encoders might.) Since we want
to be able to dequantize all the components at the end of the file, this
means that we have to save away the table actually used for each component.
We do this by copying the table at the start of the first scan containing
the component.
The JPEG spec prohibits the encoder from changing the contents of a Q-table
slot between scans of a component using that slot. If the encoder does so
anyway, this decoder will simply use the Q-table values that were current
at the start of the first scan for the component.

* The decompressor output side looks only at the saved quant tables,
* not at the current Q-table slots.
*/

```

```

LOCAL(void)
latch_quant_tables (j_decompress_ptr cinfo)
{
    int ci, qtblno;
    jpeg_component_info *compptr;
    JQUANT_TBL * qtbl;

    for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
        compptr = cinfo->cur_comp_info[ci];
        /* No work if we already saved Q-table for this component */
        if (compptr->quant_table != NULL)
            continue;
        /* Make sure specified quantization table is present */
        qtblno = compptr->quant_tbl_no;
        if (qtblno < 0 || qtblno >= NUM_QUANT_TBLS ||
            cinfo->quant_tbl_ptrs[qtblno] == NULL)
            ERREXIT1(cinfo, JERR_NO_QUANT_TABLE, qtblno);
        /* OK, save away the quantization table */
        qtbl = (JQUANT_TBL *)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                SIZEOF(JQUANT_TBL));
        MEMCOPY(qtbl, cinfo->quant_tbl_ptrs[qtblno], SIZEOF(JQUANT_TBL));
    }
}

```



```

    compptr->quant_table = qtbl;
}
}

/*
 * Initialize the input modules to read a scan of compressed data.
 * The first call to this is done by jdmaster.c after initializing
 * the entire decompressor (during jpeg_start_decompress).
 * Subsequent calls come from consume_markers, below.
 */

METHODDEF(void)
start_input_pass (j_decompress_ptr cinfo)
{
    per_scan_setup(cinfo);
    latch_quant_tables(cinfo);
    (*cinfo->entropy->start_pass) (cinfo);
    (*cinfo->coef->start_input_pass) (cinfo);
    cinfo->inputctl->consume_input = cinfo->coef->consume_data;
}

/*
 * Finish up after inputting a compressed-data scan.
 * This is called by the coefficient controller after it's read all
 * the expected data of the scan.
 */

METHODDEF(void)
finish_input_pass (j_decompress_ptr cinfo)
{
    cinfo->inputctl->consume_input = consume_markers;
}

/*
 * Read JPEG markers before, between, or after compressed-data scans.
 * Change state as necessary when a new scan is reached.
 * Return value is JPEG_SUSPENDED, JPEG_REACHED_SOS, or JPEG_REACHED_EOI.
 *
 * The consume_input method pointer points either here or to the
 * coefficient controller's consume_data routine, depending on whether
 * we are reading a compressed data segment or inter-segment markers.
 */

METHODDEF(int)
consume_markers (j_decompress_ptr cinfo)
{
    my_inputctl_ptr inputctl = (my_inputctl_ptr) cinfo->inputctl;
    int val;

    if (inputctl->pub.eoi_reached) /* After hitting EOI, read no further */
        return JPEG_REACHED_EOI;

    val = (*cinfo->marker->read_markers) (cinfo);

    switch (val) {
        case JPEG_REACHED_SOS: /* Found SOS */
            if (inputctl->inheaders) { /* 1st SOS */
                initial_setup(cinfo);
                inputctl->inheaders = FALSE;
                /* Note: start_input_pass must be called by jdmaster.c
                 * before any more input can be consumed. jdapimin.c is
                 * responsible for enforcing this sequencing.
                 */
            } else { /* 2nd or later SOS marker */
                if (! inputctl->pub.has_multiple_scans)
                    ERREXIT(cinfo, JERR_EOI_EXPECTED); /* Oops, I wasn't expecting this! */
                start_input_pass(cinfo);
            }
            break;
        case JPEG_REACHED_EOI: /* Found EOI */
            inputctl->pub.eoi_reached = TRUE;
            if (inputctl->inheaders) { /* Tables-only datastream, apparently */
                if (cinfo->marker->saw_SOF)
                    ERREXIT(cinfo, JERR_SOF_NO_SOS);
            } else {
                /* Prevent infinite loop in coef ctlr's decompress_data routine
                 * if user set output_scan_number larger than number of scans.

```

```

        */
        if (cinfo->output_scan_number > cinfo->input_scan_number)
            cinfo->output_scan_number = cinfo->input_scan_number;
    }
    break;
case JPEG_SUSPENDED:
    break;
}

return val;
}

/*
 * Reset state to begin a fresh datastream.
 */

METHODDEF(void)
reset_input_controller (j_decompress_ptr cinfo)
{
    my_inputctl_ptr inputctl = (my_inputctl_ptr) cinfo->inputctl;

    inputctl->pub.consume_input = consume_markers;
    inputctl->pub.has_multiple_scans = FALSE; /* "unknown" would be better */
    inputctl->pub.eoi_reached = FALSE;
    inputctl->inheaders = TRUE;
    /* Reset other modules */
    (*cinfo->err->reset_error_mgr) ((j_common_ptr) cinfo);
    (*cinfo->marker->reset_marker_reader) (cinfo);
    /* Reset progression state -- would be cleaner if entropy decoder did this */
    cinfo->coef_bits = NULL;
}

/*
 * Initialize the input controller module.
 * This is called only once, when the decompression object is created.
 */

GLOBAL(void)
jinit_input_controller (j_decompress_ptr cinfo)
{
    my_inputctl_ptr inputctl;

    /* Create subobject in permanent pool */
    inputctl = (my_inputctl_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_PERMANENT,
            SIZEOF(my_input_controller));
    cinfo->inputctl = (struct jpeg_input_controller *) inputctl;
    /* Initialize method pointers */
    inputctl->pub.consume_input = consume_markers;
    inputctl->pub.reset_input_controller = reset_input_controller;
    inputctl->pub.start_input_pass = start_input_pass;
    inputctl->pub.finish_input_pass = finish_input_pass;
    /* Initialize state: can't use reset_input_controller since we don't
     * want to try to reset other modules yet.
     */
    inputctl->pub.has_multiple_scans = FALSE; /* "unknown" would be better */
    inputctl->pub.eoi_reached = FALSE;
    inputctl->inheaders = TRUE;
}

```

```

/*
 * jdmainct.c
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains the main buffer controller for decompression.
 * The main buffer lies between the JPEG decompressor proper and the
 * post-processor; it holds downsampled data in the JPEG colorspace.
 *
 * Note that this code is bypassed in raw-data mode, since the application
 * supplies the equivalent of the main buffer in that case.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/*
 * In the current system design, the main buffer need never be a full-image
 * buffer; any full-height buffers will be found inside the coefficient or
 * postprocessing controllers. Nonetheless, the main controller is not
 * trivial. Its responsibility is to provide context rows for upsampling/
 * rescaling, and doing this in an efficient fashion is a bit tricky.
 *
 * Postprocessor input data is counted in "row groups". A row group
 * is defined to be (v_samp_factor * DCT_scaled_size / min_DCT_scaled_size)
 * sample rows of each component. (We require DCT_scaled_size values to be
 * chosen such that these numbers are integers. In practice DCT_scaled_size
 * values will likely be powers of two, so we actually have the stronger
 * condition that DCT_scaled_size / min_DCT_scaled_size is an integer.)
 * Upsampling will typically produce max_v_samp_factor pixel rows from each
 * row group (times any additional scale factor that the upsampler is
 * applying).
 *
 * The coefficient controller will deliver data to us one iMCU row at a time;
 * each iMCU row contains v_samp_factor * DCT_scaled_size sample rows, or
 * exactly min_DCT_scaled_size row groups. (This amount of data corresponds
 * to one row of MCUs when the image is fully interleaved.) Note that the
 * number of sample rows varies across components, but the number of row
 * groups does not. Some garbage sample rows may be included in the last iMCU
 * row at the bottom of the image.
 *
 * Depending on the vertical scaling algorithm used, the upsampler may need
 * access to the sample row(s) above and below its current input row group.
 * The upsampler is required to set need_context_rows TRUE at global selection
 * time if so. When need_context_rows is FALSE, this controller can simply
 * obtain one iMCU row at a time from the coefficient controller and dole it
 * out as row groups to the postprocessor.
 *
 * When need_context_rows is TRUE, this controller guarantees that the buffer
 * passed to postprocessing contains at least one row group's worth of samples
 * above and below the row group(s) being processed. Note that the context
 * rows "above" the first passed row group appear at negative row offsets in
 * the passed buffer. At the top and bottom of the image, the required
 * context rows are manufactured by duplicating the first or last real sample
 * row; this avoids having special cases in the upsampling inner loops.
 *
 * The amount of context is fixed at one row group just because that's a
 * convenient number for this controller to work with. The existing
 * upsamplers really only need one sample row of context. An upsampler
 * supporting arbitrary output rescaling might wish for more than one row
 * group of context when shrinking the image; tough, we don't handle that.
 * (This is justified by the assumption that downsizing will be handled mostly
 * by adjusting the DCT_scaled_size values, so that the actual scale factor at
 * the upsample step needn't be much less than one.)
 *
 * To provide the desired context, we have to retain the last two row groups
 * of one iMCU row while reading in the next iMCU row. (The last row group
 * can't be processed until we have another row group for its below-context,
 * and so we have to save the next-to-last group too for its above-context.)
 * We could do this most simply by copying data around in our buffer, but
 * that'd be very slow. We can avoid copying any data by creating a rather
 * strange pointer structure. Here's how it works. We allocate a workspace
 * consisting of M+2 row groups (where M = min_DCT_scaled_size is the number
 * of row groups per iMCU row). We create two sets of redundant pointers to
 * the workspace. Labeling the physical row groups 0 to M+1, the synthesized
 * pointer lists look like this:

```

```

*           M+1           M-1
* master pointer --> 0      master pointer --> 0
*           1           1
*           ...           ...
*           M-3           M-3
*           M-2           M
*           M-1           M+1
*           M             M-2
*           M+1           M-1
*           0             0
* We read alternate iMCU rows using each master pointer; thus the last two
* row groups of the previous iMCU row remain un-overwritten in the workspace.
* The pointer lists are set up so that the required context rows appear to
* be adjacent to the proper places when we pass the pointer lists to the
* upsampler.
*
* The above pictures describe the normal state of the pointer lists.
* At top and bottom of the image, we diddle the pointer lists to duplicate
* the first or last sample row as necessary (this is cheaper than copying
* sample rows around).
*
* This scheme breaks down if M < 2, ie, min_DCT_scaled_size is 1. In that
* situation each iMCU row provides only one row group so the buffering logic
* must be different (eg, we must read two iMCU rows before we can emit the
* first row group). For now, we simply do not support providing context
* rows when min_DCT_scaled_size is 1. That combination seems unlikely to
* be worth providing --- if someone wants a 1/8th-size preview, they probably
* want it quick and dirty, so a context-free upsampler is sufficient.
*/

```

```

/* Private buffer controller object */

```

```

typedef struct {
    struct jpeg_d_main_controller pub; /* public fields */

    /* Pointer to allocated workspace (M or M+2 row groups). */
    JSAMPARRAY buffer[MAX_COMPONENTS];

    boolean buffer_full; /* Have we gotten an iMCU row from decoder? */
    JDIMENSION rowgroup_ctr; /* counts row groups output to postprocessor */

    /* Remaining fields are only used in the context case. */

    /* These are the master pointers to the funny-order pointer lists. */
    JSAMPIMAGE xbuffer[2]; /* pointers to weird pointer lists */

    int whichptr; /* indicates which pointer set is now in use */
    int context_state; /* process_data state machine status */
    JDIMENSION rowgroups_avail; /* row groups available to postprocessor */
    JDIMENSION iMCU_row_ctr; /* counts iMCU rows to detect image top/bot */
    my_main_controller;
} my_main_controller;

typedef my_main_controller * my_main_ptr;

```

```

/* context_state values: */
#define CTX_PREPARE_FOR_IMCU 0 /* need to prepare for MCU row */
#define CTX_PROCESS_IMCU 1 /* feeding iMCU to postprocessor */
#define CTX_POSTPONED_ROW 2 /* feeding postponed row group */

```

```

/* Forward declarations */

```

```

METHODDEF(void) process_data_simple_main
    JPP((j_decompress_ptr cinfo, JSAMPARRAY output_buf,
         JDIMENSION *out_row_ctr, JDIMENSION out_rows_avail));
METHODDEF(void) process_data_context_main
    JPP((j_decompress_ptr cinfo, JSAMPARRAY output_buf,
         JDIMENSION *out_row_ctr, JDIMENSION out_rows_avail));
#ifdef QUANT_2PASS_SUPPORTED
METHODDEF(void) process_data_crank_post
    JPP((j_decompress_ptr cinfo, JSAMPARRAY output_buf,
         JDIMENSION *out_row_ctr, JDIMENSION out_rows_avail));
#endif

```

```

LOCAL(void)
alloc_funny_pointers(j_decompress_ptr cinfo)
/* Allocate space for the funny pointer lists.
 * This is done only once, not once per pass.
 */

```

```

{
my_main_ptr main = (my_main_ptr) cinfo->main;
int ci, rgroup;
int M = cinfo->min_DCT_scaled_size;
jpeg_component_info *comp_ptr;
JSAMPARRAY xbuf;

/* Get top-level space for component array pointers.
 * We alloc both arrays with one call to save a few cycles.
 */
main->xbuffer[0] = (JSAMPIMAGE)
    (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
        cinfo->num_components * 2 * sizeof(JSAMPARRAY));
main->xbuffer[1] = main->xbuffer[0] + cinfo->num_components;

for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
    ci++, comp_ptr++) {
    rgroup = (comp_ptr->v_samp_factor * comp_ptr->DCT_scaled_size) /
        cinfo->min_DCT_scaled_size; /* height of a row group of component */
    /* Get space for pointer lists --- M+4 row groups in each list.
     * We alloc both pointer lists with one call to save a few cycles.
     */
    xbuf = (JSAMPARRAY)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            2 * (rgroup * (M + 4)) * sizeof(JSAMPARRAY));
    xbuf += rgroup; /* want one row group at negative offsets */
    main->xbuffer[0][ci] = xbuf;
    xbuf += rgroup * (M + 4);
    main->xbuffer[1][ci] = xbuf;
}
}

LOCAL(void)
make_funny_pointers (j_decompress_ptr cinfo)
/* Create the funny pointer lists discussed in the comments above.
 * The actual workspace is already allocated (in main->buffer),
 * and the space for the pointer lists is allocated too.
 * This routine just fills in the curiously ordered lists.
 * This will be repeated at the beginning of each pass.
 */
{
my_main_ptr main = (my_main_ptr) cinfo->main;
int ci, i, rgroup;
int M = cinfo->min_DCT_scaled_size;
jpeg_component_info *comp_ptr;
JSAMPARRAY buf, xbuf0, xbuf1;

for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
    ci++, comp_ptr++) {
    rgroup = (comp_ptr->v_samp_factor * comp_ptr->DCT_scaled_size) /
        cinfo->min_DCT_scaled_size; /* height of a row group of component */
    xbuf0 = main->xbuffer[0][ci];
    xbuf1 = main->xbuffer[1][ci];
    /* First copy the workspace pointers as-is */
    buf = main->buffer[ci];
    for (i = 0; i < rgroup * (M + 2); i++) {
        xbuf0[i] = xbuf1[i] = buf[i];
    }
    /* In the second list, put the last four row groups in swapped order */
    for (i = 0; i < rgroup * 2; i++) {
        xbuf1[rgroup*(M-2) + i] = buf[rgroup*M + i];
        xbuf1[rgroup*M + i] = buf[rgroup*(M-2) + i];
    }
    /* The wraparound pointers at top and bottom will be filled later
     * (see set_wraparound_pointers, below). Initially we want the "above"
     * pointers to duplicate the first actual data line. This only needs
     * to happen in xbuffer[0].
     */
    for (i = 0; i < rgroup; i++) {
        xbuf0[i - rgroup] = xbuf0[0];
    }
}
}

LOCAL(void)
set_wraparound_pointers (j_decompress_ptr cinfo)
/* Set up the "wraparound" pointers at top and bottom of the pointer lists.
 * This changes the pointer list state from top-of-image to the normal state.
 */

```

```

*/
{
    my_main_ptr main = (my_main_ptr) cinfo->main;
    int ci, i, rgroup;
    int M = cinfo->min_DCT_scaled_size;
    jpeg_component_info *comp_ptr;
    JSAMPARRAY xbuf0, xbuf1;

    for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
         ci++, comp_ptr++) {
        rgroup = (comp_ptr->v_samp_factor * comp_ptr->DCT_scaled_size) /
            cinfo->min_DCT_scaled_size; /* height of a row group of component */
        xbuf0 = main->xbuffer[0][ci];
        xbuf1 = main->xbuffer[1][ci];
        for (i = 0; i < rgroup; i++) {
            xbuf0[i - rgroup] = xbuf0[rgroup*(M+1) + i];
            xbuf1[i - rgroup] = xbuf1[rgroup*(M+1) + i];
            xbuf0[rgroup*(M+2) + i] = xbuf0[i];
            xbuf1[rgroup*(M+2) + i] = xbuf1[i];
        }
    }
}

LOCAL(void)
set_bottom_pointers (j_decompress_ptr cinfo)
/* Change the pointer lists to duplicate the last sample row at the bottom
 * of the image.  which_ptr indicates which xbuffer holds the final iMCU row.
 * Also sets rowgroups_avail to indicate number of nondummy row groups in row.
 */
{
    my_main_ptr main = (my_main_ptr) cinfo->main;
    int ci, i, rgroup, iMCUheight, rows_left;
    jpeg_component_info *comp_ptr;
    JSAMPARRAY xbuf;

    for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
         ci++, comp_ptr++) {
        /* Count sample rows in one iMCU row and in one row group */
        iMCUheight = comp_ptr->v_samp_factor * comp_ptr->DCT_scaled_size;
        rgroup = iMCUheight / cinfo->min_DCT_scaled_size;
        /* Count nondummy sample rows remaining for this component */
        rows_left = (int) (comp_ptr->downsampled_height % (JDIMENSION) iMCUheight);
        if (rows_left == 0) rows_left = iMCUheight;
        /* Count nondummy row groups.  Should get same answer for each component,
         * so we need only do it once.
         */
        if (ci == 0) {
            main->rowgroups_avail = (JDIMENSION) ((rows_left-1) / rgroup + 1);
        }
        /* Duplicate the last real sample row rgroup*2 times; this pads out the
         * last partial rowgroup and ensures at least one full rowgroup of context.
         */
        xbuf = main->xbuffer[main->which_ptr][ci];
        for (i = 0; i < rgroup * 2; i++) {
            xbuf[rows_left + i] = xbuf[rows_left-1];
        }
    }
}

/*
 * Initialize for a processing pass.
 */

METHODDEF(void)
start_pass_main (j_decompress_ptr cinfo, J_BUF_MODE pass_mode)
{
    my_main_ptr main = (my_main_ptr) cinfo->main;

    switch (pass_mode) {
        case JBUF_PASS_THRU:
            if (cinfo->upsample->need_context_rows) {
                main->pub.process_data = process_data_context_main;
                make_funny_pointers(cinfo); /* Create the xbuffer[] lists */
                main->which_ptr = 0; /* Read first iMCU row into xbuffer[0] */
                main->context_state = CTX_PREPARE_FOR_IMCU;
                main->iMCU_row_ctr = 0;
            } else {
                /* Simple case with no context needed */
            }
    }
}

```

```

    main->pub.process_data = process_data_simple_main;
}
main->buffer_full = FALSE; /* Mark buffer empty */
main->rowgroup_ctr = 0;
break;
#endif QUANT_2PASS_SUPPORTED
case JBUF_CRANK_DEST:
/* For last pass of 2-pass quantization, just crank the postprocessor */
main->pub.process_data = process_data_crank_post;
break;
#endif
default:
ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
break;
}
}

/*
 * Process some data.
 * This handles the simple case where no context is required.
 */

METHODDEF(void)
process_data_simple_main (j_decompress_ptr cinfo,
                          JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
                          JDIMENSION out_rows_avail)
{
    my_main_ptr main = (my_main_ptr) cinfo->main;
    JDIMENSION rowgroups_avail;

    /* Read input data if we haven't filled the main buffer yet */
    if (! main->buffer_full) {
        if (! (*cinfo->coef->decompress_data) (cinfo, main->buffer))
            return; /* suspension forced, can do nothing more */
        main->buffer_full = TRUE; /* OK, we have an iMCU row to work with */
    }

    /* There are always min_DCT_scaled_size row groups in an iMCU row. */
    rowgroups_avail = (JDIMENSION) cinfo->min_DCT_scaled_size;
    /* Note: at the bottom of the image, we may pass extra garbage row groups
     * to the postprocessor. The postprocessor has to check for bottom
     * of image anyway (at row resolution), so no point in us doing it too.
     */

    /* Feed the postprocessor */
    (*cinfo->post->post_process_data) (cinfo, main->buffer,
                                       &main->rowgroup_ctr, rowgroups_avail,
                                       output_buf, out_row_ctr, out_rows_avail);

    /* Has postprocessor consumed all the data yet? If so, mark buffer empty */
    if (main->rowgroup_ctr >= rowgroups_avail) {
        main->buffer_full = FALSE;
        main->rowgroup_ctr = 0;
    }
}

/*
 * Process some data.
 * This handles the case where context rows must be provided.
 */

METHODDEF(void)
process_data_context_main (j_decompress_ptr cinfo,
                           JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
                           JDIMENSION out_rows_avail)
{
    my_main_ptr main = (my_main_ptr) cinfo->main;

    /* Read input data if we haven't filled the main buffer yet */
    if (! main->buffer_full) {
        if (! (*cinfo->coef->decompress_data) (cinfo,
                                              main->xbuffer[main->whichptr]))
            return; /* suspension forced, can do nothing more */
        main->buffer_full = TRUE; /* OK, we have an iMCU row to work with */
        main->iMCU_row_ctr++; /* count rows received */
    }

    /* Postprocessor typically will not swallow all the input data it is handed

```

```

* in one call (due to filling the output buffer first). Must be prepared
* to exit and restart. This switch lets us keep track of how far we got.
* Note that each case falls through to the next on successful completion.
*/
switch (main->context_state) {
case CTX_POSTPONED_ROW:
/* Call postprocessor using previously set pointers for postponed row */
(*cinfo->post->post_process_data) (cinfo, main->xbuffer[main->whichptr],
&main->rowgroup_ctr, main->rowgroups_avail,
output_buf, out_row_ctr, out_rows_avail);
if (main->rowgroup_ctr < main->rowgroups_avail)
return; /* Need to suspend */
main->context_state = CTX_PREPARE_FOR_IMCU;
if (*out_row_ctr >= out_rows_avail)
return; /* Postprocessor exactly filled output buf */
/*FALLTHROUGH*/
case CTX_PREPARE_FOR_IMCU:
/* Prepare to process first M-1 row groups of this iMCU row */
main->rowgroup_ctr = 0;
main->rowgroups_avail = (JDIMENSION) (cinfo->min_DCT_scaled_size - 1);
/* Check for bottom of image: if so, tweak pointers to "duplicate"
* the last sample row, and adjust rowgroups_avail to ignore padding rows.
*/
if (main->iMCU_row_ctr == cinfo->total_iMCU_rows)
set_bottom_pointers(cinfo);
main->context_state = CTX_PROCESS_IMCU;
/*FALLTHROUGH*/
case CTX_PROCESS_IMCU:
/* Call postprocessor using previously set pointers */
(*cinfo->post->post_process_data) (cinfo, main->xbuffer[main->whichptr],
&main->rowgroup_ctr, main->rowgroups_avail,
output_buf, out_row_ctr, out_rows_avail);
if (main->rowgroup_ctr < main->rowgroups_avail)
return; /* Need to suspend */
/* After the first iMCU, change wraparound pointers to normal state */
if (main->iMCU_row_ctr == 1)
set_wraparound_pointers(cinfo);
/* Prepare to load new iMCU row using other xbuffer list */
main->whichptr ^= 1; /* 0=>1 or 1=>0 */
main->buffer_full = FALSE;
/* Still need to process last row group of this iMCU row, */
/* which is saved at index M+1 of the other xbuffer */
main->rowgroup_ctr = (JDIMENSION) (cinfo->min_DCT_scaled_size + 1);
main->rowgroups_avail = (JDIMENSION) (cinfo->min_DCT_scaled_size + 2);
main->context_state = CTX_POSTPONED_ROW;
}
}

/*
Process some data.
Final pass of two-pass quantization: just call the postprocessor.
Source data will be the postprocessor controller's internal buffer.
*/

#ifdef QUANT_2PASS_SUPPORTED

METHODDEF(void)
process_data_crank_post (j_decompress_ptr cinfo,
JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
JDIMENSION out_rows_avail)
{
(*cinfo->post->post_process_data) (cinfo, (JSAMPIMAGE) NULL,
(JDIMENSION *) NULL, (JDIMENSION) 0,
output_buf, out_row_ctr, out_rows_avail);
}

#endif /* QUANT_2PASS_SUPPORTED */

/*
* Initialize main buffer controller.
*/

GLOBAL(void)
jinit_d_main_controller (j_decompress_ptr cinfo, boolean need_full_buffer)
{
my_main_ptr main;
int ci, rgroup, ngroups;
jpeg_component_info *comp_ptr;

```



```

main = (my_main_ptr)
    (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
        SIZEOF(my_main_controller));
cinfo->main = (struct jpeg_d_main_controller *) main;
main->pub.start_pass = start_pass_main;

if (need_full_buffer) /* shouldn't happen */
    ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);

/* Allocate the workspace.
 * ngroups is the number of row groups we need.
 */
if (cinfo->upsample->need_context_rows) {
    if (cinfo->min_DCT_scaled_size < 2) /* unsupported, see comments above */
        ERREXIT(cinfo, JERR_NOTIMPL);
    alloc_funny_pointers(cinfo); /* Alloc space for xbuffer[] lists */
    ngroups = cinfo->min_DCT_scaled_size + 2;
} else {
    ngroups = cinfo->min_DCT_scaled_size;
}

for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
    ci++, compptr++) {
    rgroup = (compptr->v_samp_factor * compptr->DCT_scaled_size) /
        cinfo->min_DCT_scaled_size; /* height of a row group of component */
    main->buffer[ci] = (*cinfo->mem->alloc_sarray)
        ((j_common_ptr) cinfo, JPOOL_IMAGE,
        compptr->width_in_blocks * compptr->DCT_scaled_size,
        (JDIMENSION) (rgroup * ngroups));
}
}

```

```

/*
 * jdmarker.c
 *
 * Copyright (C) 1991-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains routines to decode JPEG datastream markers.
 * Most of the complexity arises from our desire to support input
 * suspension: if not all of the data for a marker is available,
 * we must exit back to the application.  On resumption, we reprocess
 * the marker.
 */

```

```

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

```

```

typedef enum {                /* JPEG marker codes */
  M_SOF0  = 0xc0,
  M_SOF1  = 0xc1,
  M_SOF2  = 0xc2,
  M_SOF3  = 0xc3,

  M_SOF5  = 0xc5,
  M_SOF6  = 0xc6,
  M_SOF7  = 0xc7,

  M_JPG   = 0xc8,
  M_SOF9  = 0xc9,
  M_SOF10 = 0xca,
  M_SOF11 = 0xcb,

  M_SOF13 = 0xcd,
  M_SOF14 = 0xce,
  M_SOF15 = 0xcf,

  M_DHT   = 0xc4,

  M_DAC   = 0xcc,

  M_RST0  = 0xd0,
  M_RST1  = 0xd1,
  M_RST2  = 0xd2,
  M_RST3  = 0xd3,
  M_RST4  = 0xd4,
  M_RST5  = 0xd5,
  M_RST6  = 0xd6,
  M_RST7  = 0xd7,

  M_SOI   = 0xd8,
  M_EOI   = 0xd9,
  M_SOS   = 0xda,
  M_DQT   = 0xdb,
  M_DNL   = 0xdc,
  M_DRI   = 0xdd,
  M_DHP   = 0xde,
  M_EXP   = 0xdf,

  M_APP0  = 0xe0,
  M_APP1  = 0xe1,
  M_APP2  = 0xe2,
  M_APP3  = 0xe3,
  M_APP4  = 0xe4,
  M_APP5  = 0xe5,
  M_APP6  = 0xe6,
  M_APP7  = 0xe7,
  M_APP8  = 0xe8,
  M_APP9  = 0xe9,
  M_APP10 = 0xea,
  M_APP11 = 0xeb,
  M_APP12 = 0xec,
  M_APP13 = 0xed,
  M_APP14 = 0xee,
  M_APP15 = 0xef,

  M_JPG0  = 0xf0,
  M_JPG13 = 0xfd,
  M_COM   = 0xfe,

```

```

M_TEM    = 0x01,

M_ERROR  = 0x100
} JPEG_MARKER;

/* Private state */

typedef struct {
    struct jpeg_marker_reader pub; /* public fields */

    /* Application-overridable marker processing methods */
    jpeg_marker_parser_method process_COM;
    jpeg_marker_parser_method process_APPn[16];

    /* Limit on marker data length to save for each marker type */
    unsigned int length_limit_COM;
    unsigned int length_limit_APPn[16];

    /* Status of COM/APPn marker saving */
    jpeg_saved_marker_ptr cur_marker; /* NULL if not processing a marker */
    unsigned int bytes_read; /* data bytes read so far in marker */
    /* Note: cur_marker is not linked into marker_list until it's all read. */
} my_marker_reader;

typedef my_marker_reader * my_marker_ptr;

/*
 * Macros for fetching data from the data source module.
 *
 * At all times, cinfo->src->next_input_byte and ->bytes_in_buffer reflect
 * the current restart point; we update them only when we have reached a
 * suitable place to restart if a suspension occurs.
 */

/* Declare and initialize local copies of input pointer/count */
#define INPUT_VARS(cinfo) \
    struct jpeg_source_mgr * datasrc = (cinfo)->src; \
    const JOCTET * next_input_byte = datasrc->next_input_byte; \
    size_t bytes_in_buffer = datasrc->bytes_in_buffer

/* Unload the local copies --- do this only at a restart boundary */
#define INPUT_SYNC(cinfo) \
    ( datasrc->next_input_byte = next_input_byte, \
      datasrc->bytes_in_buffer = bytes_in_buffer )

/* Reload the local copies --- used only in MAKE_BYTE_AVAIL */
#define INPUT_RELOAD(cinfo) \
    ( next_input_byte = datasrc->next_input_byte, \
      bytes_in_buffer = datasrc->bytes_in_buffer )

/* Internal macro for INPUT_BYTE and INPUT_2BYTES: make a byte available.
 * Note we do *not* do INPUT_SYNC before calling fill_input_buffer,
 * but we must reload the local copies after a successful fill.
 */
#define MAKE_BYTE_AVAIL(cinfo,action) \
    if (bytes_in_buffer == 0) { \
        if (! (*datasrc->fill_input_buffer) (cinfo)) \
            { action; } \
        INPUT_RELOAD(cinfo); \
    }

/* Read a byte into variable V.
 * If must suspend, take the specified action (typically "return FALSE").
 */
#define INPUT_BYTE(cinfo,V,action) \
    MAKESTMT( MAKE_BYTE_AVAIL(cinfo,action); \
              bytes_in_buffer--; \
              V = GETJOCTET(*next_input_byte++); )

/* As above, but read two bytes interpreted as an unsigned 16-bit integer.
 * V should be declared unsigned int or perhaps INT32.
 */
#define INPUT_2BYTES(cinfo,V,action) \
    MAKESTMT( MAKE_BYTE_AVAIL(cinfo,action); \
              bytes_in_buffer--; \
              V = ((unsigned int) GETJOCTET(*next_input_byte++)) << 8; \
              MAKE_BYTE_AVAIL(cinfo,action); \
    )

```

```

bytes_in_buffer--; \
V += GETJOCTET(*next_input_byte++); )

```

```

/*
 * Routines to process JPEG markers.
 *
 * Entry condition: JPEG marker itself has been read and its code saved
 *   in cinfo->unread_marker; input restart point is just after the marker.
 *
 * Exit: if return TRUE, have read and processed any parameters, and have
 *   updated the restart point to point after the parameters.
 *   If return FALSE, was forced to suspend before reaching end of
 *   marker parameters; restart point has not been moved. Same routine
 *   will be called again after application supplies more input data.
 *
 * This approach to suspension assumes that all of a marker's parameters
 * can fit into a single input bufferload. This should hold for "normal"
 * markers. Some COM/APPn markers might have large parameter segments
 * that might not fit. If we are simply dropping such a marker, we use
 * skip_input_data to get past it, and thereby put the problem on the
 * source manager's shoulders. If we are saving the marker's contents
 * into memory, we use a slightly different convention: when forced to
 * suspend, the marker processor updates the restart point to the end of
 * what it's consumed (ie, the end of the buffer) before returning FALSE.
 * On resumption, cinfo->unread_marker still contains the marker code,
 * but the data source will point to the next chunk of marker data.
 * The marker processor must retain internal state to deal with this.
 *
 * Note that we don't bother to avoid duplicate trace messages if a
 * suspension occurs within marker parameters. Other side effects
 * require more care.
 */

```

```

LOCAL(boolean)
get_soi (j_decompress_ptr cinfo)
/* Process an SOI marker */
{
    int i;

    TRACEMS(cinfo, 1, JTRC_SOI);

    if (cinfo->marker->saw_SOI)
        ERREXIT(cinfo, JERR_SOI_DUPLICATE);

    /* Reset all parameters that are defined to be reset by SOI */
    for (i = 0; i < NUM_ARITH_TBLS; i++) {
        cinfo->arith_dc_L[i] = 0;
        cinfo->arith_dc_U[i] = 1;
        cinfo->arith_ac_K[i] = 5;
    }
    cinfo->restart_interval = 0;

    /* Set initial assumptions for colorspace etc */

    cinfo->jpeg_color_space = JCS_UNKNOWN;
    cinfo->CCIR601_sampling = FALSE; /* Assume non-CCIR sampling??? */

    cinfo->saw_JFIF_marker = FALSE;
    cinfo->JFIF_major_version = 1; /* set default JFIF APP0 values */
    cinfo->JFIF_minor_version = 1;
    cinfo->density_unit = 0;
    cinfo->X_density = 1;
    cinfo->Y_density = 1;
    cinfo->saw_Adobe_marker = FALSE;
    cinfo->Adobe_transform = 0;

    cinfo->marker->saw_SOI = TRUE;

    return TRUE;
}

```

```

LOCAL(boolean)
get_sof (j_decompress_ptr cinfo, boolean is_prog, boolean is_arith)
/* Process a SOFn marker */
{
    INT32 length;

```

```

int c, ci;
jpeg_component_info * compptr;
INPUT_VARS(cinfo);

cinfo->progressive_mode = is_prog;
cinfo->arith_code = is_arith;

INPUT_2BYTES(cinfo, length, return FALSE);

INPUT_BYTE(cinfo, cinfo->data_precision, return FALSE);
INPUT_2BYTES(cinfo, cinfo->image_height, return FALSE);
INPUT_2BYTES(cinfo, cinfo->image_width, return FALSE);
INPUT_BYTE(cinfo, cinfo->num_components, return FALSE);

length -= 8;

TRACEMS4(cinfo, 1, JTRC_SOF, cinfo->unread_marker,
(int) cinfo->image_width, (int) cinfo->image_height,
cinfo->num_components);

if (cinfo->marker->saw_SOF)
ERREXIT(cinfo, JERR_SOF_DUPLICATE);

/* We don't support files in which the image height is initially specified */
/* as 0 and is later redefined by DNL. As long as we have to check that, */
/* might as well have a general sanity check. */
if (cinfo->image_height <= 0 || cinfo->image_width <= 0
|| cinfo->num_components <= 0)
ERREXIT(cinfo, JERR_EMPTY_IMAGE);

if (length != (cinfo->num_components * 3))
ERREXIT(cinfo, JERR_BAD_LENGTH);

if (cinfo->comp_info == NULL) /* do only once, even if suspend */
cinfo->comp_info = (jpeg_component_info *) (*cinfo->mem->alloc_small)
((j_common_ptr) cinfo, JPOOL_IMAGE,
cinfo->num_components * SIZEOF(jpeg_component_info));

for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
ci++, compptr++) {
compptr->component_index = ci;
INPUT_BYTE(cinfo, compptr->component_id, return FALSE);
INPUT_BYTE(cinfo, c, return FALSE);
compptr->h_samp_factor = (c >> 4) & 15;
compptr->v_samp_factor = (c & 15);
INPUT_BYTE(cinfo, compptr->quant_tbl_no, return FALSE);

TRACEMS4(cinfo, 1, JTRC_SOF_COMPONENT,
compptr->component_id, compptr->h_samp_factor,
compptr->v_samp_factor, compptr->quant_tbl_no);

cinfo->marker->saw_SOF = TRUE;

INPUT_SYNC(cinfo);
return TRUE;
}

LOCAL(boolean)
get_sos (j_decompress_ptr cinfo)
/* Process a SOS marker */
{
INT32 length;
int i, ci, n, c, cc;
jpeg_component_info * compptr;
INPUT_VARS(cinfo);

if (! cinfo->marker->saw_SOF)
ERREXIT(cinfo, JERR_SOS_NO_SOF);

INPUT_2BYTES(cinfo, length, return FALSE);

INPUT_BYTE(cinfo, n, return FALSE); /* Number of components */

TRACEMS1(cinfo, 1, JTRC_SOS, n);

if (length != (n * 2 + 6) || n < 1 || n > MAX_COMPS_IN_SCAN)
ERREXIT(cinfo, JERR_BAD_LENGTH);

```

```

cinfo->comps_in_scan = n;

/* Collect the component-spec parameters */
for (i = 0; i < n; i++) {
    INPUT_BYTE(cinfo, cc, return FALSE);
    INPUT_BYTE(cinfo, c, return FALSE);

    for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
        ci++, compptr++) {
        if (cc == compptr->component_id)
            goto id_found;
    }

    ERREXIT1(cinfo, JERR_BAD_COMPONENT_ID, cc);
}

id_found:

cinfo->cur_comp_info[i] = compptr;
compptr->dc_tbl_no = (c >> 4) & 15;
compptr->ac_tbl_no = (c >> 4) & 15;

TRACEMS3(cinfo, 1, JTRC_SOS_COMPONENT, cc,
    compptr->dc_tbl_no, compptr->ac_tbl_no);
}

/* Collect the additional scan parameters Ss, Se, Ah/Al. */
INPUT_BYTE(cinfo, c, return FALSE);
cinfo->Ss = c;
INPUT_BYTE(cinfo, c, return FALSE);
cinfo->Se = c;
INPUT_BYTE(cinfo, c, return FALSE);
cinfo->Ah = (c >> 4) & 15;
cinfo->Al = (c >> 4) & 15;

TRACEMS4(cinfo, 1, JTRC_SOS_PARAMS, cinfo->Ss, cinfo->Se,
    cinfo->Ah, cinfo->Al);

/* Prepare to scan data & restart markers */
cinfo->marker->next_restart_num = 0;

/* Count another SOS marker */
cinfo->input_scan_number++;

INPUT_SYNC(cinfo);
return TRUE;
}

#ifdef D_ARITH_CODING_SUPPORTED
LOCAL(boolean)
get_dac (j_decompress_ptr cinfo)
/* Process a DAC marker */
{
    INT32 length;
    int index, val;
    INPUT_VARS(cinfo);

    INPUT_2BYTES(cinfo, length, return FALSE);
    length -= 2;

    while (length > 0) {
        INPUT_BYTE(cinfo, index, return FALSE);
        INPUT_BYTE(cinfo, val, return FALSE);

        length -= 2;

        TRACEMS2(cinfo, 1, JTRC_DAC, index, val);

        if (index < 0 || index >= (2*NUM_ARITH_TBLS))
            ERREXIT1(cinfo, JERR_DAC_INDEX, index);

        if (index >= NUM_ARITH_TBLS) { /* define AC table */
            cinfo->arith_ac_K[index-NUM_ARITH_TBLS] = (UINT8) val;
        } else { /* define DC table */
            cinfo->arith_dc_L[index] = (UINT8) (val & 0x0F);
            cinfo->arith_dc_U[index] = (UINT8) (val >> 4);
            if (cinfo->arith_dc_L[index] > cinfo->arith_dc_U[index])
                ERREXIT1(cinfo, JERR_DAC_VALUE, val);
        }
    }
}

```

```

    }
}

if (length != 0)
    ERREXIT(cinfo, JERR_BAD_LENGTH);

INPUT_SYNC(cinfo);
return TRUE;
}

#else /* ! D_ARITH_CODING_SUPPORTED */

#define get_dac(cinfo) skip_variable(cinfo)

#endif /* D_ARITH_CODING_SUPPORTED */

LOCAL(boolean)
get_dht (j_decompress_ptr cinfo)
/* Process a DHT marker */
{
    INT32 length;
    UINT8 bits[17];
    UINT8 huffval[256];
    int i, index, count;
    JHUFF_TBL **htblptr;
    INPUT_VARS(cinfo);

    INPUT_2BYTES(cinfo, length, return FALSE);
    length -= 2;

    while (length > 16) {
        INPUT_BYTE(cinfo, index, return FALSE);
        TRACEMS1(cinfo, 1, JTRC_DHT, index);

        bits[0] = 0;
        count = 0;
        for (i = 1; i <= 16; i++) {
            INPUT_BYTE(cinfo, bits[i], return FALSE);
            count += bits[i];
        }

        length -= 1 + 16;

        TRACEMS8(cinfo, 2, JTRC_HUFFBITS,
            bits[1], bits[2], bits[3], bits[4],
            bits[5], bits[6], bits[7], bits[8]);
        TRACEMS8(cinfo, 2, JTRC_HUFFBITS,
            bits[9], bits[10], bits[11], bits[12],
            bits[13], bits[14], bits[15], bits[16]);

        /* Here we just do minimal validation of the counts to avoid walking
         * off the end of our table space.  jdhuft.c will check more carefully.
         */
        if (count > 256 || ((INT32) count) > length)
            ERREXIT(cinfo, JERR_BAD_HUFF_TABLE);

        for (i = 0; i < count; i++)
            INPUT_BYTE(cinfo, huffval[i], return FALSE);

        length -= count;

        if (index & 0x10) { /* AC table definition */
            index -= 0x10;
            htblptr = &cinfo->ac_huff_tbl_ptrs[index];
        } else { /* DC table definition */
            htblptr = &cinfo->dc_huff_tbl_ptrs[index];
        }

        if (index < 0 || index >= NUM_HUFF_TBLS)
            ERREXIT1(cinfo, JERR_DHT_INDEX, index);

        if (*htblptr == NULL)
            *htblptr = jpeg_alloc_huff_table((j_common_ptr) cinfo);

        MEMCOPY((*htblptr)->bits, bits, SIZEOF((*htblptr)->bits));
        MEMCOPY((*htblptr)->huffval, huffval, SIZEOF((*htblptr)->huffval));
    }
}

```

```

    if (length != 0)
        ERREXIT(cinfo, JERR_BAD_LENGTH);

    INPUT_SYNC(cinfo);
    return TRUE;
}

LOCAL(boolean)
get_dqt (j_decompress_ptr cinfo)
/* Process a DQT marker */
{
    INT32 length;
    int n, i, prec;
    unsigned int tmp;
    JQUANT_TBL *quant_ptr;
    INPUT_VARS(cinfo);

    INPUT_2BYTES(cinfo, length, return FALSE);
    length -= 2;

    while (length > 0) {
        INPUT_BYTE(cinfo, n, return FALSE);
        prec = n >> 4;
        n &= 0x0F;

        TRACEMS2(cinfo, 1, JTRC_DQT, n, prec);

        if (n >= NUM_QUANT_TBLS)
            ERREXIT1(cinfo, JERR_DQT_INDEX, n);

        if (cinfo->quant_tbl_ptrs[n] == NULL)
            cinfo->quant_tbl_ptrs[n] = jpeg_alloc_quant_table((j_common_ptr) cinfo);
        quant_ptr = cinfo->quant_tbl_ptrs[n];

        for (i = 0; i < DCTSIZE2; i++) {
            if (prec)
                INPUT_2BYTES(cinfo, tmp, return FALSE);
            else
                INPUT_BYTE(cinfo, tmp, return FALSE);
            /* We convert the zigzag-order table to natural array order. */
            quant_ptr->quantval[jpeg_natural_order[i]] = (UINT16) tmp;
        }

        if (cinfo->err->trace_level >= 2) {
            for (i = 0; i < DCTSIZE2; i += 8) {
                TRACEMS8(cinfo, 2, JTRC_QUANTVALS,
                    quant_ptr->quantval[i], quant_ptr->quantval[i+1],
                    quant_ptr->quantval[i+2], quant_ptr->quantval[i+3],
                    quant_ptr->quantval[i+4], quant_ptr->quantval[i+5],
                    quant_ptr->quantval[i+6], quant_ptr->quantval[i+7]);
            }
        }

        length -= DCTSIZE2+1;
        if (prec) length -= DCTSIZE2;
    }

    if (length != 0)
        ERREXIT(cinfo, JERR_BAD_LENGTH);

    INPUT_SYNC(cinfo);
    return TRUE;
}

LOCAL(boolean)
get_dri (j_decompress_ptr cinfo)
/* Process a DRI marker */
{
    INT32 length;
    unsigned int tmp;
    INPUT_VARS(cinfo);

    INPUT_2BYTES(cinfo, length, return FALSE);

    if (length != 4)
        ERREXIT(cinfo, JERR_BAD_LENGTH);

    INPUT_2BYTES(cinfo, tmp, return FALSE);

```



```

TRACEMS1(cinfo, 1, JTRC_DRI, tmp);

cinfo->restart_interval = tmp;

INPUT_SYNC(cinfo);
return TRUE;
}

/*
 * Routines for processing APPn and COM markers.
 * These are either saved in memory or discarded, per application request.
 * APP0 and APP14 are specially checked to see if they are
 * JFIF and Adobe markers, respectively.
 */

#define APP0_DATA_LEN 14 /* Length of interesting data in APP0 */
#define APP14_DATA_LEN 12 /* Length of interesting data in APP14 */
#define APPN_DATA_LEN 14 /* Must be the largest of the above!! */

LOCAL(void)
examine_app0 (j_decompress_ptr cinfo, JOCTET * data,
              unsigned int datalen, INT32 remaining)
/* Examine first few bytes from an APP0.
 * Take appropriate action if it is a JFIF marker.
 * datalen is # of bytes at data[], remaining is length of rest of marker data.
 */
{
    INT32 totallen = (INT32) datalen + remaining;

    if (datalen >= APP0_DATA_LEN &&
        GETJOCTET(data[0]) == 0x4A &&
        GETJOCTET(data[1]) == 0x46 &&
        GETJOCTET(data[2]) == 0x49 &&
        GETJOCTET(data[3]) == 0x46 &&
        GETJOCTET(data[4]) == 0) {
        /* Found JFIF APP0 marker: save info */
        cinfo->saw_JFIF_marker = TRUE;
        cinfo->JFIF_major_version = GETJOCTET(data[5]);
        cinfo->JFIF_minor_version = GETJOCTET(data[6]);
        cinfo->density_unit = GETJOCTET(data[7]);
        cinfo->X_density = (GETJOCTET(data[8]) << 8) + GETJOCTET(data[9]);
        cinfo->Y_density = (GETJOCTET(data[10]) << 8) + GETJOCTET(data[11]);
        /* Check version.
         * Major version must be 1, anything else signals an incompatible change.
         * (We used to treat this as an error, but now it's a nonfatal warning,
         * because some bozo at Hijaak couldn't read the spec.)
         * Minor version should be 0..2, but process anyway if newer.
         */
        if (cinfo->JFIF_major_version != 1)
            WARNMS2(cinfo, JWRN_JFIF_MAJOR,
                  cinfo->JFIF_major_version, cinfo->JFIF_minor_version);
        /* Generate trace messages */
        TRACEMS5(cinfo, 1, JTRC_JFIF,
                cinfo->JFIF_major_version, cinfo->JFIF_minor_version,
                cinfo->X_density, cinfo->Y_density, cinfo->density_unit);
        /* Validate thumbnail dimensions and issue appropriate messages */
        if (GETJOCTET(data[12]) | GETJOCTET(data[13]))
            TRACEMS2(cinfo, 1, JTRC_JFIF_THUMBNAI,
                    GETJOCTET(data[12]), GETJOCTET(data[13]));
        totallen -= APP0_DATA_LEN;
        if (totallen !=
            ((INT32)GETJOCTET(data[12]) * (INT32)GETJOCTET(data[13]) * (INT32) 3))
            TRACEMS1(cinfo, 1, JTRC_JFIF_BADTHUMBNAISIZE, (int) totallen);
    } else if (datalen >= 6 &&
        GETJOCTET(data[0]) == 0x4A &&
        GETJOCTET(data[1]) == 0x46 &&
        GETJOCTET(data[2]) == 0x58 &&
        GETJOCTET(data[3]) == 0x58 &&
        GETJOCTET(data[4]) == 0) {
        /* Found JFIF "JFXX" extension APP0 marker */
        /* The library doesn't actually do anything with these,
         * but we try to produce a helpful trace message.
         */
        switch (GETJOCTET(data[5])) {
        case 0x10:
            TRACEMS1(cinfo, 1, JTRC_THUMB_JPEG, (int) totallen);
            break;

```

```

case 0x11:
    TRACEMS1(cinfo, 1, JTRC_THUMB_PALETTE, (int) totallen);
    break;
case 0x13:
    TRACEMS1(cinfo, 1, JTRC_THUMB_RGB, (int) totallen);
    break;
default:
    TRACEMS2(cinfo, 1, JTRC_JFIF_EXTENSION,
        GETJOCTET(data[5]), (int) totallen);
    break;
}
} else {
    /* Start of APP0 does not match "JFIF" or "JFXX", or too short */
    TRACEMS1(cinfo, 1, JTRC_APP0, (int) totallen);
}
}

LOCAL(void)
examine_app14 (j_decompress_ptr cinfo, JOCTET * data,
    unsigned int datalen, INT32 remaining)
/* Examine first few bytes from an APP14.
 * Take appropriate action if it is an Adobe marker.
 * datalen is # of bytes at data[], remaining is length of rest of marker data.
 */
{
    unsigned int version, flags0, flags1, transform;

    if (datalen >= APP14_DATA_LEN &&
        GETJOCTET(data[0]) == 0x41 &&
        GETJOCTET(data[1]) == 0x64 &&
        GETJOCTET(data[2]) == 0x6F &&
        GETJOCTET(data[3]) == 0x62 &&
        GETJOCTET(data[4]) == 0x65) {
        /* Found Adobe APP14 marker */
        version = (GETJOCTET(data[5]) << 8) + GETJOCTET(data[6]);
        flags0 = (GETJOCTET(data[7]) << 8) + GETJOCTET(data[8]);
        flags1 = (GETJOCTET(data[9]) << 8) + GETJOCTET(data[10]);
        transform = GETJOCTET(data[11]);
        TRACEMS4(cinfo, 1, JTRC_ADOBE, version, flags0, flags1, transform);
        cinfo->saw_Adobe_marker = TRUE;
        cinfo->Adobe_transform = (UINT8) transform;
    } else {
        /* Start of APP14 does not match "Adobe", or too short */
        TRACEMS1(cinfo, 1, JTRC_APP14, (int) (datalen + remaining));
    }
}

METHODDEF(boolean)
get_interesting_appn (j_decompress_ptr cinfo)
/* Process an APP0 or APP14 marker without saving it */
{
    INT32 length;
    JOCTET b[APPN_DATA_LEN];
    unsigned int i, numtoread;
    INPUT_VARS(cinfo);

    INPUT_2BYTES(cinfo, length, return FALSE);
    length -= 2;

    /* get the interesting part of the marker data */
    if (length >= APPN_DATA_LEN)
        numtoread = APPN_DATA_LEN;
    else if (length > 0)
        numtoread = (unsigned int) length;
    else
        numtoread = 0;
    for (i = 0; i < numtoread; i++)
        INPUT_BYTE(cinfo, b[i], return FALSE);
    length -= numtoread;

    /* process it */
    switch (cinfo->unread_marker) {
    case M_APP0:
        examine_app0(cinfo, (JOCTET *) b, numtoread, length);
        break;
    case M_APP14:
        examine_app14(cinfo, (JOCTET *) b, numtoread, length);
        break;
    }
}

```

[illegible]

```

default:
/* can't get here unless jpeg_save_markers chooses wrong processor */
ERREXIT1(cinfo, JERR_UNKNOWN_MARKER, cinfo->unread_marker);
break;
}

/* skip any remaining data -- could be lots */
INPUT_SYNC(cinfo);
if (length > 0)
    (*cinfo->src->skip_input_data) (cinfo, (long) length);

return TRUE;
}

#ifdef SAVE_MARKERS_SUPPORTED

METHODDEF(boolean)
save_marker (j_decompress_ptr cinfo)
/* Save an APPn or COM marker into the marker list */
{
    my_marker_ptr marker = (my_marker_ptr) cinfo->marker;
    jpeg_saved_marker_ptr cur_marker = marker->cur_marker;
    unsigned int bytes_read, data_length;
    JOCTET * data;
    INT32 length = 0;
    INPUT_VARS(cinfo);

    if (cur_marker == NULL) {
        /* begin reading a marker */
        INPUT_2BYTES(cinfo, length, return FALSE);
        length -= 2;
        if (length >= 0) { /* watch out for bogus length word */
            /* figure out how much we want to save */
            unsigned int limit;
            if (cinfo->unread_marker == (int) M_COM)
                limit = marker->length_limit_COM;
            else
                limit = marker->length_limit_APPn[cinfo->unread_marker - (int) M_APP0];
            if ((unsigned int) length < limit)
                limit = (unsigned int) length;
            /* allocate and initialize the marker item */
            cur_marker = (jpeg_saved_marker_ptr)
                (*cinfo->mem->alloc_large) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                    SIZEOF(struct jpeg_marker_struct) + limit);
            cur_marker->next = NULL;
            cur_marker->marker = (UINT8) cinfo->unread_marker;
            cur_marker->original_length = (unsigned int) length;
            cur_marker->data_length = limit;
            /* data area is just beyond the jpeg_marker_struct */
            data = cur_marker->data = (JOCTET *) (cur_marker + 1);
            marker->cur_marker = cur_marker;
            marker->bytes_read = 0;
            bytes_read = 0;
            data_length = limit;
        } else {
            /* deal with bogus length word */
            bytes_read = data_length = 0;
            data = NULL;
        }
    } else {
        /* resume reading a marker */
        bytes_read = marker->bytes_read;
        data_length = cur_marker->data_length;
        data = cur_marker->data + bytes_read;
    }

    while (bytes_read < data_length) {
        INPUT_SYNC(cinfo); /* move the restart point to here */
        marker->bytes_read = bytes_read;
        /* If there's not at least one byte in buffer, suspend */
        MAKE_BYTE_AVAIL(cinfo, return FALSE);
        /* Copy bytes with reasonable rapidity */
        while (bytes_read < data_length && bytes_in_buffer > 0) {
            *data++ = *next_input_byte++;
            bytes_in_buffer--;
            bytes_read++;
        }
    }
}

```

```

/* Done reading what we want to read */
if (cur_marker != NULL) { /* will be NULL if bogus length word */
/* Add new marker to end of list */
if (cinfo->marker_list == NULL) {
cinfo->marker_list = cur_marker;
} else {
jpeg_saved_marker_ptr prev = cinfo->marker_list;
while (prev->next != NULL)
prev = prev->next;
prev->next = cur_marker;
}
/* Reset pointer & calc remaining data length */
data = cur_marker->data;
length = cur_marker->original_length - data_length;
}
/* Reset to initial state for next marker */
marker->cur_marker = NULL;

/* Process the marker if interesting; else just make a generic trace msg */
switch (cinfo->unread_marker) {
case M_APP0:
examine_app0(cinfo, data, data_length, length);
break;
case M_APP14:
examine_app14(cinfo, data, data_length, length);
break;
default:
TRACEMS2(cinfo, 1, JTRC_MISC_MARKER, cinfo->unread_marker,
(int) (data_length + length));
break;
}
}
/* skip any remaining data -- could be lots */
INPUT_SYNC(cinfo); /* do before skip_input_data */
if (length > 0)
(*cinfo->src->skip_input_data) (cinfo, (long) length);
return TRUE;
}
#endif /* SAVE_MARKERS_SUPPORTED */

METHODDEF(boolean)
skip_variable (j_decompress_ptr cinfo)
/* Skip over an unknown or uninteresting variable-length marker */
{
INT32 length;
INPUT_VARS(cinfo);
INPUT_2BYTES(cinfo, length, return FALSE);
length -= 2;
TRACEMS2(cinfo, 1, JTRC_MISC_MARKER, cinfo->unread_marker, (int) length);
INPUT_SYNC(cinfo); /* do before skip_input_data */
if (length > 0)
(*cinfo->src->skip_input_data) (cinfo, (long) length);
return TRUE;
}

/*
* Find the next JPEG marker, save it in cinfo->unread_marker.
* Returns FALSE if had to suspend before reaching a marker;
* in that case cinfo->unread_marker is unchanged.
*
* Note that the result might not be a valid marker code,
* but it will never be 0 or FF.
*/
LOCAL(boolean)
next_marker (j_decompress_ptr cinfo)
{
int c;
INPUT_VARS(cinfo);
for (;;) {
INPUT_BYTE(cinfo, c, return FALSE);

```

```

/* Skip any non-FF bytes.
 * This may look a bit inefficient, but it will not occur in a valid file.
 * We sync after each discarded byte so that a suspending data source
 * can discard the byte from its buffer.
 */
while (c != 0xFF) {
    cinfo->marker->discarded_bytes++;
    INPUT_SYNC(cinfo);
    INPUT_BYTE(cinfo, c, return FALSE);
}
/* This loop swallows any duplicate FF bytes. Extra FFs are legal as
 * pad bytes, so don't count them in discarded_bytes. We assume there
 * will not be so many consecutive FF bytes as to overflow a suspending
 * data source's input buffer.
 */
do {
    INPUT_BYTE(cinfo, c, return FALSE);
} while (c == 0xFF);
if (c != 0)
    break; /* found a valid marker, exit loop */
/* Reach here if we found a stuffed-zero data sequence (FF/00).
 * Discard it and loop back to try again.
 */
cinfo->marker->discarded_bytes += 2;
INPUT_SYNC(cinfo);
}

if (cinfo->marker->discarded_bytes != 0) {
    WARNMS2(cinfo, JWRN_EXTRANEEOUS_DATA, cinfo->marker->discarded_bytes, c);
    cinfo->marker->discarded_bytes = 0;
}

cinfo->unread_marker = c;
INPUT_SYNC(cinfo);
return TRUE;
}

LOCAL(boolean)
first_marker (j_decompress_ptr cinfo)
/* Like next_marker, but used to obtain the initial SOI marker. */
/* For this marker, we do not allow preceding garbage or fill; otherwise,
 * we might well scan an entire input file before realizing it ain't JPEG.
 * If an application wants to process non-JFIF files, it must seek to the
 * SOI before calling the JPEG library.
 */
{
    int c, c2;
    INPUT_VARS(cinfo);

    INPUT_BYTE(cinfo, c, return FALSE);
    INPUT_BYTE(cinfo, c2, return FALSE);
    if (c != 0xFF || c2 != (int) M_SOI)
        ERREXIT2(cinfo, JERR_NO_SOI, c, c2);

    cinfo->unread_marker = c2;

    INPUT_SYNC(cinfo);
    return TRUE;
}

/*
 * Read markers until SOS or EOI.
 *
 * Returns same codes as are defined for jpeg_consume_input:
 * JPEG_SUSPENDED, JPEG_REACHED_SOS, or JPEG_REACHED_EOI.
 */

METHODDEF(int)
read_markers (j_decompress_ptr cinfo)
{
    /* Outer loop repeats once for each marker. */
    for (;;) {
        /* Collect the marker proper, unless we already did. */
        /* NB: first_marker() enforces the requirement that SOI appear first. */
        if (cinfo->unread_marker == 0) {
            if (! cinfo->marker->saw_SOI) {
                if (! first_marker(cinfo))

```

```

    return JPEG_SUSPENDED;
} else {
if (! next_marker(cinfo))
    return JPEG_SUSPENDED;
}
}
/* At this point cinfo->unread_marker contains the marker code and the
 * input point is just past the marker proper, but before any parameters.
 * A suspension will cause us to return with this state still true.
 */
switch (cinfo->unread_marker) {
case M_SOI:
    if (! get_soi(cinfo))
        return JPEG_SUSPENDED;
    break;

case M_SOF0:          /* Baseline */
case M_SOF1:          /* Extended sequential, Huffman */
    if (! get_sof(cinfo, FALSE, FALSE))
        return JPEG_SUSPENDED;
    break;

case M_SOF2:          /* Progressive, Huffman */
    if (! get_sof(cinfo, TRUE, FALSE))
        return JPEG_SUSPENDED;
    break;

case M_SOF9:          /* Extended sequential, arithmetic */
    if (! get_sof(cinfo, FALSE, TRUE))
        return JPEG_SUSPENDED;
    break;

case M_SOF10:         /* Progressive, arithmetic */
    if (! get_sof(cinfo, TRUE, TRUE))
        return JPEG_SUSPENDED;
    break;

/* Currently unsupported SOFn types */
case M_SOF3:          /* Lossless, Huffman */
case M_SOF5:          /* Differential sequential, Huffman */
case M_SOF6:          /* Differential progressive, Huffman */
case M_SOF7:          /* Differential lossless, Huffman */
case M_JPG:           /* Reserved for JPEG extensions */
case M_SOF11:         /* Lossless, arithmetic */
case M_SOF13:         /* Differential sequential, arithmetic */
case M_SOF14:         /* Differential progressive, arithmetic */
case M_SOF15:         /* Differential lossless, arithmetic */
    ERREXIT1(cinfo, JERR_SOF_UNSUPPORTED, cinfo->unread_marker);
    break;

case M_SOS:
    if (! get_sos(cinfo))
        return JPEG_SUSPENDED;
    cinfo->unread_marker = 0; /* processed the marker */
    return JPEG_REACHED_SOS;

case M_EOI:
    TRACEMS(cinfo, 1, JTRC_EOI);
    cinfo->unread_marker = 0; /* processed the marker */
    return JPEG_REACHED_EOI;

case M_DAC:
    if (! get_dac(cinfo))
        return JPEG_SUSPENDED;
    break;

case M_DHT:
    if (! get_dht(cinfo))
        return JPEG_SUSPENDED;
    break;

case M_DQT:
    if (! get_dqt(cinfo))
        return JPEG_SUSPENDED;
    break;

case M_DRI:
    if (! get_dri(cinfo))
        return JPEG_SUSPENDED;
    break;

```

```

case M_APP0:
case M_APP1:
case M_APP2:
case M_APP3:
case M_APP4:
case M_APP5:
case M_APP6:
case M_APP7:
case M_APP8:
case M_APP9:
case M_APP10:
case M_APP11:
case M_APP12:
case M_APP13:
case M_APP14:
case M_APP15:
    if (! ((my_marker_ptr) cinfo->marker)->process_APPn[
        cinfo->unread_marker - (int) M_APP0]) (cinfo))
return JPEG_SUSPENDED;
    break;

case M_COM:
    if (! ((my_marker_ptr) cinfo->marker)->process_COM) (cinfo))
return JPEG_SUSPENDED;
    break;

case M_RST0:          /* these are all parameterless */
case M_RST1:
case M_RST2:
case M_RST3:
case M_RST4:
case M_RST5:
case M_RST6:
case M_RST7:
case M_TEM:
    TRACEMS1(cinfo, 1, JTRC_PARMLESS_MARKER, cinfo->unread_marker);
    break;

case M_DNL:           /* Ignore DNL ... perhaps the wrong thing */
    if (! skip_variable(cinfo))
return JPEG_SUSPENDED;
    break;

default:              /* must be DHP, EXP, JPGn, or RESn */
    /* For now, we treat the reserved markers as fatal errors since they are
     * likely to be used to signal incompatible JPEG Part 3 extensions.
     * Once the JPEG 3 version-number marker is well defined, this code
     * ought to change!
     */
    ERREXIT1(cinfo, JERR_UNKNOWN_MARKER, cinfo->unread_marker);
    break;
}
/* Successfully processed marker, so reset state variable */
cinfo->unread_marker = 0;
} /* end loop */
}

/*
 * Read a restart marker, which is expected to appear next in the datastream;
 * if the marker is not there, take appropriate recovery action.
 * Returns FALSE if suspension is required.
 *
 * This is called by the entropy decoder after it has read an appropriate
 * number of MCUs.  cinfo->unread_marker may be nonzero if the entropy decoder
 * has already read a marker from the data source.  Under normal conditions
 * cinfo->unread_marker will be reset to 0 before returning; if not reset,
 * it holds a marker which the decoder will be unable to read past.
 */

METHODDEF(boolean)
read_restart_marker (j_decompress_ptr cinfo)
{
    /* Obtain a marker unless we already did. */
    /* Note that next_marker will complain if it skips any data. */
    if (cinfo->unread_marker == 0) {
        if (! next_marker(cinfo))
            return FALSE;
    }
}

```



```

if (cinfo->unread_marker ==
    ((int) M_RST0 + cinfo->marker->next_restart_num)) {
    /* Normal case --- swallow the marker and let entropy decoder continue */
    TRACEMS1(cinfo, 3, JTRC_RST, cinfo->marker->next_restart_num);
    cinfo->unread_marker = 0;
} else {
    /* Uh-oh, the restart markers have been messed up. */
    /* Let the data source manager determine how to resync. */
    if (! (*cinfo->src->resync_to_restart) (cinfo,
        cinfo->marker->next_restart_num))
        return FALSE;
}

/* Update next-restart state */
cinfo->marker->next_restart_num = (cinfo->marker->next_restart_num + 1) & 7;

return TRUE;
}

/*
 * This is the default resync_to_restart method for data source managers
 * to use if they don't have any better approach. Some data source managers
 * may be able to back up, or may have additional knowledge about the data
 * which permits a more intelligent recovery strategy; such managers would
 * presumably supply their own resync method.
 *
 * read_restart_marker calls resync_to_restart if it finds a marker other than
 * the restart marker it was expecting. (This code is *not* used unless
 * a nonzero restart interval has been declared.) cinfo->unread_marker is
 * the marker code actually found (might be anything, except 0 or FF).
 * The desired restart marker number (0..7) is passed as a parameter.
 * This routine is supposed to apply whatever error recovery strategy seems
 * appropriate in order to position the input stream to the next data segment.
 * Note that cinfo->unread_marker is treated as a marker appearing before
 * the current data-source input point; usually it should be reset to zero
 * before returning.
 * Returns FALSE if suspension is required.
 *
 * This implementation is substantially constrained by wanting to treat the
 * input as a data stream; this means we can't back up. Therefore, we have
 * only the following actions to work with:
 *
 * 1. Simply discard the marker and let the entropy decoder resume at next
 *    byte of file.
 *
 * 2. Read forward until we find another marker, discarding intervening
 *    data. (In theory we could look ahead within the current bufferload,
 *    without having to discard data if we don't find the desired marker.
 *    This idea is not implemented here, in part because it makes behavior
 *    dependent on buffer size and chance buffer-boundary positions.)
 *
 * 3. Leave the marker unread (by failing to zero cinfo->unread_marker).
 *    This will cause the entropy decoder to process an empty data segment,
 *    inserting dummy zeroes, and then we will reprocess the marker.
 *
 * #2 is appropriate if we think the desired marker lies ahead, while #3 is
 * appropriate if the found marker is a future restart marker (indicating
 * that we have missed the desired restart marker, probably because it got
 * corrupted).
 * We apply #2 or #3 if the found marker is a restart marker no more than
 * two counts behind or ahead of the expected one. We also apply #2 if the
 * found marker is not a legal JPEG marker code (it's certainly bogus data).
 * If the found marker is a restart marker more than 2 counts away, we do #1
 * (too much risk that the marker is erroneous; with luck we will be able to
 * resync at some future point).
 * For any valid non-restart JPEG marker, we apply #3. This keeps us from
 * overrunning the end of a scan. An implementation limited to single-scan
 * files might find it better to apply #2 for markers other than EOI, since
 * any other marker would have to be bogus data in that case.
 */

GLOBAL(boolean)
jpeg_resync_to_restart (j_decompress_ptr cinfo, int desired)
{
    int marker = cinfo->unread_marker;
    int action = 1;

    /* Always put up a warning. */
    WARNMS2(cinfo, JWRN_MUST_RESYNC, marker, desired);

    /* Outer loop handles repeated decision after scanning forward. */

```

```

for (;;) {
    if (marker < (int) M_SOF0)
        action = 2; /* invalid marker */
    else if (marker < (int) M_RST0 || marker > (int) M_RST7)
        action = 3; /* valid non-restart marker */
    else {
        if (marker == ((int) M_RST0 + ((desired+1) & 7)) ||
            marker == ((int) M_RST0 + ((desired+2) & 7)))
            action = 3; /* one of the next two expected restarts */
        else if (marker == ((int) M_RST0 + ((desired-1) & 7)) ||
            marker == ((int) M_RST0 + ((desired-2) & 7)))
            action = 2; /* a prior restart, so advance */
        else
            action = 1; /* desired restart or too far away */
    }
    TRACEMS2(cinfo, 4, JTRC_RECOVERY_ACTION, marker, action);
    switch (action) {
    case 1:
        /* Discard marker and let entropy decoder resume processing. */
        cinfo->unread_marker = 0;
        return TRUE;
    case 2:
        /* Scan to the next marker, and repeat the decision loop. */
        if (! next_marker(cinfo))
            return FALSE;
        marker = cinfo->unread_marker;
        break;
    case 3:
        /* Return without advancing past this marker. */
        /* Entropy decoder will be forced to process an empty segment. */
        return TRUE;
    }
} /* end loop */

```

/\* Reset marker processing state to begin a fresh datastream.

```

METHODDEF(void)
reset_marker_reader (j_decompress_ptr cinfo)
{
    my_marker_ptr marker = (my_marker_ptr) cinfo->marker;

    cinfo->comp_info = NULL; /* until allocated by get_sof */
    cinfo->input_scan_number = 0; /* no SOS seen yet */
    cinfo->unread_marker = 0; /* no pending marker */
    marker->pub.saw_SOI = FALSE; /* set internal state too */
    marker->pub.saw_SOF = FALSE;
    marker->pub.discarded_bytes = 0;
    marker->cur_marker = NULL;
}

```

```

/*
 * Initialize the marker reader module.
 * This is called only once, when the decompression object is created.
 */

```

```

GLOBAL(void)
jinit_marker_reader (j_decompress_ptr cinfo)
{
    my_marker_ptr marker;
    int i;

    /* Create subobject in permanent pool */
    marker = (my_marker_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_PERMANENT,
            SIZEOF(my_marker_reader));
    cinfo->marker = (struct jpeg_marker_reader *) marker;
    /* Initialize public method pointers */
    marker->pub.reset_marker_reader = reset_marker_reader;
    marker->pub.read_markers = read_markers;
    marker->pub.read_restart_marker = read_restart_marker;
    /* Initialize COM/APPn processing.
     * By default, we examine and then discard APP0 and APP14,
     * but simply discard COM and all other APPn.
     */
    marker->process_COM = skip_variable;
}

```

```

marker->length_limit_COM = 0;
for (i = 0; i < 16; i++) {
    marker->process_APPn[i] = skip_variable;
    marker->length_limit_APPn[i] = 0;
}
marker->process_APPn[0] = get_interesting_appn;
marker->process_APPn[14] = get_interesting_appn;
/* Reset marker processing state */
reset_marker_reader(cinfo);
}

/*
 * Control saving of COM and APPn markers into marker_list.
 */

#ifdef SAVE_MARKERS_SUPPORTED

GLOBAL(void)
jpeg_save_markers (j_decompress_ptr cinfo, int marker_code,
                   unsigned int length_limit)
{
    my_marker_ptr marker = (my_marker_ptr) cinfo->marker;
    long maxlength;
    jpeg_marker_parser_method processor;

    /* Length limit mustn't be larger than what we can allocate
     * (should only be a concern in a 16-bit environment).
     */
    maxlength = cinfo->mem->max_alloc_chunk - SIZEOF(struct jpeg_marker_struct);
    if (((long) length_limit) > maxlength)
        length_limit = (unsigned int) maxlength;

    /* Choose processor routine to use.
     * APP0/APP14 have special requirements.
     */
    if (length_limit) {
        processor = save_marker;
        /* If saving APP0/APP14, save at least enough for our internal use. */
        if (marker_code == (int) M_APP0 && length_limit < APP0_DATA_LEN)
            length_limit = APP0_DATA_LEN;
        else if (marker_code == (int) M_APP14 && length_limit < APP14_DATA_LEN)
            length_limit = APP14_DATA_LEN;
    } else {
        processor = skip_variable;
        /* If discarding APP0/APP14, use our regular on-the-fly processor. */
        if (marker_code == (int) M_APP0 || marker_code == (int) M_APP14)
            processor = get_interesting_appn;
    }

    if (marker_code == (int) M_COM) {
        marker->process_COM = processor;
        marker->length_limit_COM = length_limit;
    } else if (marker_code >= (int) M_APP0 && marker_code <= (int) M_APP15) {
        marker->process_APPn[marker_code - (int) M_APP0] = processor;
        marker->length_limit_APPn[marker_code - (int) M_APP0] = length_limit;
    } else
        ERREXIT1(cinfo, JERR_UNKNOWN_MARKER, marker_code);
}

#endif /* SAVE_MARKERS_SUPPORTED */

/*
 * Install a special processing method for COM or APPn markers.
 */

GLOBAL(void)
jpeg_set_marker_processor (j_decompress_ptr cinfo, int marker_code,
                           jpeg_marker_parser_method routine)
{
    my_marker_ptr marker = (my_marker_ptr) cinfo->marker;

    if (marker_code == (int) M_COM)
        marker->process_COM = routine;
    else if (marker_code >= (int) M_APP0 && marker_code <= (int) M_APP15)
        marker->process_APPn[marker_code - (int) M_APP0] = routine;
    else
        ERREXIT1(cinfo, JERR_UNKNOWN_MARKER, marker_code);
}

```



```

/*
 * jdmaster.c
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains master control logic for the JPEG decompressor.
 * These routines are concerned with selecting the modules to be executed
 * and with determining the number of passes and the work to be done in each
 * pass.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/* Private state */

typedef struct {
  struct jpeg_decomp_master pub; /* public fields */

  int pass_number; /* # of passes completed */

  boolean using_merged_upsample; /* TRUE if using merged upsample/cconvert */

  /* Saved references to initialized quantizer modules,
   * in case we need to switch modes.
   */
  struct jpeg_color_quantizer * quantizer_1pass;
  struct jpeg_color_quantizer * quantizer_2pass;
} my_decomp_master;

typedef my_decomp_master * my_master_ptr;

/*
 * Determine whether merged upsample/color conversion should be used.
 * CRUCIAL: this must match the actual capabilities of jdmerge.c!
 */
LOCAL(boolean)
use_merged_upsample (j_decompress_ptr cinfo)
{
#ifdef UPSAMPLE_MERGING_SUPPORTED
  /* Merging is the equivalent of plain box-filter upsampling */
  if (cinfo->do_fancy_upsampling || cinfo->CCIR601_sampling)
    return FALSE;
  /* jdmerge.c only supports YCC=>RGB color conversion */
  if (cinfo->jpeg_color_space != JCS_YCbCr || cinfo->num_components != 3 ||
      cinfo->out_color_space != JCS_RGB ||
      cinfo->out_color_components != RGB_PIXELSIZE)
    return FALSE;
  /* and it only handles 2h1v or 2h2v sampling ratios */
  if (cinfo->comp_info[0].h_samp_factor != 2 ||
      cinfo->comp_info[1].h_samp_factor != 1 ||
      cinfo->comp_info[2].h_samp_factor != 1 ||
      cinfo->comp_info[0].v_samp_factor > 2 ||
      cinfo->comp_info[1].v_samp_factor != 1 ||
      cinfo->comp_info[2].v_samp_factor != 1)
    return FALSE;
  /* furthermore, it doesn't work if we've scaled the IDCTs differently */
  if (cinfo->comp_info[0].DCT_scaled_size != cinfo->min_DCT_scaled_size ||
      cinfo->comp_info[1].DCT_scaled_size != cinfo->min_DCT_scaled_size ||
      cinfo->comp_info[2].DCT_scaled_size != cinfo->min_DCT_scaled_size)
    return FALSE;
  /* ??? also need to test for upsample-time rescaling, when & if supported */
  return TRUE; /* by golly, it'll work... */
#else
  return FALSE;
#endif
}

/*
 * Compute output image dimensions and related values.
 * NOTE: this is exported for possible use by application.
 * Hence it mustn't do anything that can't be done twice.
 * Also note that it may be called before the master module is initialized!
 */

```

```

*/

GLOBAL(void)
jpeg_calc_output_dimensions (j_decompress_ptr cinfo)
/* Do computations that are needed before master selection phase */
{
#ifdef IDCT_SCALING_SUPPORTED
    int ci;
    jpeg_component_info *comp_ptr;
#endif

    /* Prevent application from calling me at wrong times */
    if (cinfo->global_state != DSTATE_READY)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);

#ifdef IDCT_SCALING_SUPPORTED

    /* Compute actual output image dimensions and DCT scaling choices. */
    if (cinfo->scale_num * 8 <= cinfo->scale_denom) {
        /* Provide 1/8 scaling */
        cinfo->output_width = (JDIMENSION)
            jdiv_round_up((long) cinfo->image_width, 8L);
        cinfo->output_height = (JDIMENSION)
            jdiv_round_up((long) cinfo->image_height, 8L);
        cinfo->min_DCT_scaled_size = 1;
    } else if (cinfo->scale_num * 4 <= cinfo->scale_denom) {
        /* Provide 1/4 scaling */
        cinfo->output_width = (JDIMENSION)
            jdiv_round_up((long) cinfo->image_width, 4L);
        cinfo->output_height = (JDIMENSION)
            jdiv_round_up((long) cinfo->image_height, 4L);
        cinfo->min_DCT_scaled_size = 2;
    } else if (cinfo->scale_num * 2 <= cinfo->scale_denom) {
        /* Provide 1/2 scaling */
        cinfo->output_width = (JDIMENSION)
            jdiv_round_up((long) cinfo->image_width, 2L);
        cinfo->output_height = (JDIMENSION)
            jdiv_round_up((long) cinfo->image_height, 2L);
        cinfo->min_DCT_scaled_size = 4;
    } else {
        /* Provide 1/1 scaling */
        cinfo->output_width = cinfo->image_width;
        cinfo->output_height = cinfo->image_height;
        cinfo->min_DCT_scaled_size = DCTSIZE;
    }

    /* In selecting the actual DCT scaling for each component, we try to
     * scale up the chroma components via IDCT scaling rather than upsampling.
     * This saves time if the upsampler gets to use 1:1 scaling.
     * Note this code assumes that the supported DCT scalings are powers of 2.
     */
    for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
         ci++, comp_ptr++) {
        int ssize = cinfo->min_DCT_scaled_size;
        while (ssize < DCTSIZE &&
            (comp_ptr->h_samp_factor * ssize * 2 <=
             cinfo->max_h_samp_factor * cinfo->min_DCT_scaled_size) &&
            (comp_ptr->v_samp_factor * ssize * 2 <=
             cinfo->max_v_samp_factor * cinfo->min_DCT_scaled_size)) {
            ssize = ssize * 2;
        }
        comp_ptr->DCT_scaled_size = ssize;
    }

    /* Recompute downsampled dimensions of components;
     * application needs to know these if using raw downsampled data.
     */
    for (ci = 0, comp_ptr = cinfo->comp_info; ci < cinfo->num_components;
         ci++, comp_ptr++) {
        /* Size in samples, after IDCT scaling */
        comp_ptr->downsampled_width = (JDIMENSION)
            jdiv_round_up((long) cinfo->image_width *
                (long) (comp_ptr->h_samp_factor * comp_ptr->DCT_scaled_size),
                (long) (cinfo->max_h_samp_factor * DCTSIZE));
        comp_ptr->downsampled_height = (JDIMENSION)
            jdiv_round_up((long) cinfo->image_height *
                (long) (comp_ptr->v_samp_factor * comp_ptr->DCT_scaled_size),
                (long) (cinfo->max_v_samp_factor * DCTSIZE));
    }

#else /* !IDCT_SCALING_SUPPORTED */

```

```

/* Hardwire it to "no scaling" */
cinfo->output_width = cinfo->image_width;
cinfo->output_height = cinfo->image_height;
/* jdinput.c has already initialized DCT_scaled_size to DCTSIZE,
 * and has computed unscaled downsampled_width and downsampled_height.
 */

#endif /* IDCT_SCALING_SUPPORTED */

/* Report number of components in selected colorspace. */
/* Probably this should be in the color conversion module... */
switch (cinfo->out_color_space) {
case JCS_GRAYSCALE:
    cinfo->out_color_components = 1;
    break;
case JCS_RGB:
    #if RGB_PIXELSIZE != 3
        cinfo->out_color_components = RGB_PIXELSIZE;
        break;
    #endif /* else share code with YCbCr */
case JCS_YCbCr:
    cinfo->out_color_components = 3;
    break;
case JCS_CMYK:
case JCS_YCCK:
    cinfo->out_color_components = 4;
    break;
default:
    /* else must be same colorspace as in file */
    cinfo->out_color_components = cinfo->num_components;
    break;
}
cinfo->output_components = (cinfo->quantize_colors ? 1 :
    cinfo->out_color_components);

/* See if upsampler will want to emit more than one row at a time */
if (use_merged_upsample(cinfo))
    cinfo->rec_outbuf_height = cinfo->max_v_samp_factor;
else
    cinfo->rec_outbuf_height = 1;

/*
 * Several decompression processes need to range-limit values to the range
 * 0..MAXJSAMPLE; the input value may fall somewhat outside this range
 * due to noise introduced by quantization, roundoff error, etc. These
 * processes are inner loops and need to be as fast as possible. On most
 * machines, particularly CPUs with pipelines or instruction prefetch,
 * a (subscript-check-less) C table lookup
 *     x = sample_range_limit[x];
 * is faster than explicit tests
 *     if (x < 0) x = 0;
 *     else if (x > MAXJSAMPLE) x = MAXJSAMPLE;
 * These processes all use a common table prepared by the routine below.
 *
 * For most steps we can mathematically guarantee that the initial value
 * of x is within MAXJSAMPLE+1 of the legal range, so a table running from
 * -(MAXJSAMPLE+1) to 2*MAXJSAMPLE+1 is sufficient. But for the initial
 * limiting step (just after the IDCT), a wildly out-of-range value is
 * possible if the input data is corrupt. To avoid any chance of indexing
 * off the end of memory and getting a bad-pointer trap, we perform the
 * post-IDCT limiting thus:
 *     x = range_limit[x & MASK];
 * where MASK is 2 bits wider than legal sample data, ie 10 bits for 8-bit
 * samples. Under normal circumstances this is more than enough range and
 * a correct output will be generated; with bogus input data the mask will
 * cause wraparound, and we will safely generate a bogus-but-in-range output.
 * For the post-IDCT step, we want to convert the data from signed to unsigned
 * representation by adding CENTERJSAMPLE at the same time that we limit it.
 * So the post-IDCT limiting table ends up looking like this:
 *     CENTERJSAMPLE, CENTERJSAMPLE+1, ..., MAXJSAMPLE,
 *     MAXJSAMPLE (repeat 2*(MAXJSAMPLE+1)-CENTERJSAMPLE times),
 *     0 (repeat 2*(MAXJSAMPLE+1)-CENTERJSAMPLE times),
 *     0, 1, ..., CENTERJSAMPLE-1
 * Negative inputs select values from the upper half of the table after
 * masking.
 *
 * We can save some space by overlapping the start of the post-IDCT table
 * with the simpler range limiting table. The post-IDCT table begins at

```

```

* sample_range_limit + CENTERJSAMPLE.
*
* Note that the table is allocated in near data space on PCs; it's small
* enough and used often enough to justify this.
*/

```

```

LOCAL(void)
prepare_range_limit_table (j_decompress_ptr cinfo)
/* Allocate and fill in the sample_range_limit table */
{
    JSAMPLE * table;
    int i;

    table = (JSAMPLE *)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            (5 * (MAXJSAMPLE+1) + CENTERJSAMPLE) * SIZEOF(JSAMPLE));
    table += (MAXJSAMPLE+1); /* allow negative subscripts of simple table */
    cinfo->sample_range_limit = table;
    /* First segment of "simple" table: limit[x] = 0 for x < 0 */
    MEMZERO(table - (MAXJSAMPLE+1), (MAXJSAMPLE+1) * SIZEOF(JSAMPLE));
    /* Main part of "simple" table: limit[x] = x */
    for (i = 0; i <= MAXJSAMPLE; i++)
        table[i] = (JSAMPLE) i;
    table += CENTERJSAMPLE; /* Point to where post-IDCT table starts */
    /* End of simple table, rest of first half of post-IDCT table */
    for (i = CENTERJSAMPLE; i < 2*(MAXJSAMPLE+1); i++)
        table[i] = MAXJSAMPLE;
    /* Second half of post-IDCT table */
    MEMZERO(table + (2 * (MAXJSAMPLE+1)),
        (2 * (MAXJSAMPLE+1) - CENTERJSAMPLE) * SIZEOF(JSAMPLE));
    MEMCOPY(table + (4 * (MAXJSAMPLE+1) - CENTERJSAMPLE),
        cinfo->sample_range_limit, CENTERJSAMPLE * SIZEOF(JSAMPLE));
}

```

```

/* Master selection of decompression modules.
* This is done once at jpeg_start_decompress time. We determine
* which modules will be used and give them appropriate initialization calls.
* We also initialize the decompressor input side to begin consuming data.
*
* Since jpeg_read_header has finished, we know what is in the SOF
* and (first) SOS markers. We also have all the application parameter
* settings.
*/

```

```

LOCAL(void)
master_selection (j_decompress_ptr cinfo)
{
    my_master_ptr master = (my_master_ptr) cinfo->master;
    boolean use_c_buffer;
    long samplesperrow;
    JDIMENSION jd_samplesperrow;

    /* Initialize dimensions and other stuff */
    jpeg_calc_output_dimensions(cinfo);
    prepare_range_limit_table(cinfo);

    /* Width of an output scanline must be representable as JDIMENSION. */
    samplesperrow = (long) cinfo->output_width * (long) cinfo->out_color_components;
    jd_samplesperrow = (JDIMENSION) samplesperrow;
    if ((long) jd_samplesperrow != samplesperrow)
        ERREXIT(cinfo, JERR_WIDTH_OVERFLOW);

    /* Initialize my private state */
    master->pass_number = 0;
    master->using_merged_upsample = use_merged_upsample(cinfo);

    /* Color quantizer selection */
    master->quantizer_1pass = NULL;
    master->quantizer_2pass = NULL;
    /* No mode changes if not using buffered-image mode. */
    if (! cinfo->quantize_colors || ! cinfo->buffered_image) {
        cinfo->enable_1pass_quant = FALSE;
        cinfo->enable_external_quant = FALSE;
        cinfo->enable_2pass_quant = FALSE;
    }
    if (cinfo->quantize_colors) {
        if (cinfo->raw_data_out)
            ERREXIT(cinfo, JERR_NOTIMPL);
    }
}

```



```

/* 2-pass quantizer only works in 3-component color space. */
if (cinfo->out_color_components != 3) {
    cinfo->enable_1pass_quant = TRUE;
    cinfo->enable_external_quant = FALSE;
    cinfo->enable_2pass_quant = FALSE;
    cinfo->colormap = NULL;
} else if (cinfo->colormap != NULL) {
    cinfo->enable_external_quant = TRUE;
} else if (cinfo->two_pass_quantize) {
    cinfo->enable_2pass_quant = TRUE;
} else {
    cinfo->enable_1pass_quant = TRUE;
}

if (cinfo->enable_1pass_quant) {
#ifdef QUANT_1PASS_SUPPORTED
    jinit_1pass_quantizer(cinfo);
    master->quantizer_1pass = cinfo->cquantize;
#else
    ERREXIT(cinfo, JERR_NOT_COMPILED);
#endif
}

/* We use the 2-pass code to map to external colormaps. */
if (cinfo->enable_2pass_quant || cinfo->enable_external_quant) {
#ifdef QUANT_2PASS_SUPPORTED
    jinit_2pass_quantizer(cinfo);
    master->quantizer_2pass = cinfo->cquantize;
#else
    ERREXIT(cinfo, JERR_NOT_COMPILED);
#endif
}

/* If both quantizers are initialized, the 2-pass one is left active;
 * this is necessary for starting with quantization to an external map.
 */

/* Post-processing: in particular, color conversion first */
if (! cinfo->raw_data_out) {
    if (master->using_merged_upsample) {
#ifdef UPSAMPLE_MERGING_SUPPORTED
        jinit_merged_upsampler(cinfo); /* does color conversion too */
    #else
        ERREXIT(cinfo, JERR_NOT_COMPILED);
    #endif
    } else {
        jinit_color_deconverter(cinfo);
        jinit_upsampler(cinfo);
    }
    jinit_d_post_controller(cinfo, cinfo->enable_2pass_quant);
}

/* Inverse DCT */
jinit_inverse_dct(cinfo);
/* Entropy decoding: either Huffman or arithmetic coding. */
if (cinfo->arith_code) {
    ERREXIT(cinfo, JERR_ARITH_NOTIMPL);
} else {
    if (cinfo->progressive_mode) {
#ifdef D_PROGRESSIVE_SUPPORTED
        jinit_phuff_decoder(cinfo);
    #else
        ERREXIT(cinfo, JERR_NOT_COMPILED);
    #endif
    } else {
        jinit_huff_decoder(cinfo);
    }
}

/* Initialize principal buffer controllers. */
use_c_buffer = cinfo->inputctl->has_multiple_scans || cinfo->buffered_image;
jinit_d_coef_controller(cinfo, use_c_buffer);

if (! cinfo->raw_data_out)
    jinit_d_main_controller(cinfo, FALSE /* never need full buffer here */);

/* We can now tell the memory manager to allocate virtual arrays. */
(*cinfo->mem->realize_virt_arrays) ((j_common_ptr) cinfo);

/* Initialize input side of decompressor to consume first scan. */
(*cinfo->inputctl->start_input_pass) (cinfo);

```

```

#ifdef D_MULTISCAN_FILES_SUPPORTED
/* If jpeg_start_decompress will read the whole file, initialize
 * progress monitoring appropriately. The input step is counted
 * as one pass.
 */
if (cinfo->progress != NULL && ! cinfo->buffered_image &&
    cinfo->inputctl->has_multiple_scans) {
    int nscans;
    /* Estimate number of scans to set pass_limit. */
    if (cinfo->progressive_mode) {
        /* Arbitrarily estimate 2 interleaved DC scans + 3 AC scans/component. */
        nscans = 2 + 3 * cinfo->num_components;
    } else {
        /* For a nonprogressive multiscan file, estimate 1 scan per component. */
        nscans = cinfo->num_components;
    }
    cinfo->progress->pass_counter = 0L;
    cinfo->progress->pass_limit = (long) cinfo->total_iMCU_rows * nscans;
    cinfo->progress->completed_passes = 0;
    cinfo->progress->total_passes = (cinfo->enable_2pass_quant ? 3 : 2);
    /* Count the input pass as done */
    master->pass_number++;
}
#endif /* D_MULTISCAN_FILES_SUPPORTED */
}

/*
 * Per-pass setup.
 * This is called at the beginning of each output pass. We determine which
 * modules will be active during this pass and give them appropriate
 * start_pass calls. We also set is_dummy_pass to indicate whether this
 * is a "real" output pass or a dummy pass for color quantization.
 * (In the latter case, jdapistd.c will crank the pass to completion.)
 */

METHODDEF(void)
prepare_for_output_pass (j_decompress_ptr cinfo)
{
    my_master_ptr master = (my_master_ptr) cinfo->master;

    if (master->pub.is_dummy_pass) {
#ifdef QUANT_2PASS_SUPPORTED
        /* Final pass of 2-pass quantization */
        master->pub.is_dummy_pass = FALSE;
        (*cinfo->cquantize->start_pass) (cinfo, FALSE);
        (*cinfo->post->start_pass) (cinfo, JBUF_CRANK_DEST);
        (*cinfo->main->start_pass) (cinfo, JBUF_CRANK_DEST);
    #else
        ERREXIT(cinfo, JERR_NOT_COMPILED);
    #endif /* QUANT_2PASS_SUPPORTED */
    } else {
        if (cinfo->quantize_colors && cinfo->colormap == NULL) {
            /* Select new quantization method */
            if (cinfo->two_pass_quantize && cinfo->enable_2pass_quant) {
                cinfo->cquantize = master->quantizer_2pass;
                master->pub.is_dummy_pass = TRUE;
            } else if (cinfo->enable_1pass_quant) {
                cinfo->cquantize = master->quantizer_1pass;
            } else {
                ERREXIT(cinfo, JERR_MODE_CHANGE);
            }
        }
        (*cinfo->idct->start_pass) (cinfo);
        (*cinfo->coef->start_output_pass) (cinfo);
        if (! cinfo->raw_data_out) {
            if (! master->using_merged_upsample)
                (*cinfo->cconvert->start_pass) (cinfo);
            (*cinfo->upsample->start_pass) (cinfo);
            if (cinfo->quantize_colors)
                (*cinfo->cquantize->start_pass) (cinfo, master->pub.is_dummy_pass);
            (*cinfo->post->start_pass) (cinfo,
                (master->pub.is_dummy_pass ? JBUF_SAVE_AND_PASS : JBUF_PASS_THRU));
            (*cinfo->main->start_pass) (cinfo, JBUF_PASS_THRU);
        }
    }

    /* Set up progress monitor's pass info if present */
    if (cinfo->progress != NULL) {
        cinfo->progress->completed_passes = master->pass_number;
    }
}

```

```

    cinfo->progress->total_passes = master->pass_number +
        (master->pub.is_dummy_pass ? 2 : 1);
/* In buffered-image mode, we assume one more output pass if EOI not
 * yet reached, but no more passes if EOI has been reached.
 */
    if (cinfo->buffered_image && ! cinfo->inputctl->eoi_reached) {
        cinfo->progress->total_passes += (cinfo->enable_2pass_quant ? 2 : 1);
    }
}
}

```

```

/*
 * Finish up at end of an output pass.
 */

```

```

METHODDEF(void)
finish_output_pass (j_decompress_ptr cinfo)
{
    my_master_ptr master = (my_master_ptr) cinfo->master;

    if (cinfo->quantize_colors)
        (*cinfo->cquantize->finish_pass) (cinfo);
    master->pass_number++;
}

```

```

#ifdef D_MULTISCAN_FILES_SUPPORTED

```

```

/*
 * Switch to a new external colormap between output passes.
 */

```

```

GLOBAL(void)
jpeg_new_colormap (j_decompress_ptr cinfo)
{
    my_master_ptr master = (my_master_ptr) cinfo->master;

/* Prevent application from calling me at wrong times */
    if (cinfo->global_state != DSTATE_BUFIMAGE)
        ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);

    if (cinfo->quantize_colors && cinfo->enable_external_quant &&
        cinfo->colormap != NULL) {
        /* Select 2-pass quantizer for external colormap use */
        cinfo->cquantize = master->quantizer_2pass;
        /* Notify quantizer of colormap change */
        (*cinfo->cquantize->new_color_map) (cinfo);
        master->pub.is_dummy_pass = FALSE; /* just in case */
    }
    else
        ERREXIT(cinfo, JERR_MODE_CHANGE);
}

```

```

#endif /* D_MULTISCAN_FILES_SUPPORTED */

```

```

/*
 * Initialize master decompression control and select active modules.
 * This is performed at the start of jpeg_start_decompress.
 */

```

```

GLOBAL(void)
jinit_master_decompress (j_decompress_ptr cinfo)
{
    my_master_ptr master;

    master = (my_master_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            sizeof(my_decomp_master));
    cinfo->master = (struct jpeg_decomp_master *) master;
    master->pub.prepare_for_output_pass = prepare_for_output_pass;
    master->pub.finish_output_pass = finish_output_pass;

    master->pub.is_dummy_pass = FALSE;

    master_selection(cinfo);
}

```

```

/*
 * jdmerge.c
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains code for merged upsampling/color conversion.
 *
 * This file combines functions from jdsample.c and jdcolor.c;
 * read those files first to understand what's going on.
 *
 * When the chroma components are to be upsampled by simple replication
 * (ie, box filtering), we can save some work in color conversion by
 * calculating all the output pixels corresponding to a pair of chroma
 * samples at one time. In the conversion equations
 *   R = Y + K1 * Cr
 *   G = Y + K2 * Cb + K3 * Cr
 *   B = Y + K4 * Cb
 * only the Y term varies among the group of pixels corresponding to a pair
 * of chroma samples, so the rest of the terms can be calculated just once.
 * At typical sampling ratios, this eliminates half or three-quarters of the
 * multiplications needed for color conversion.
 *
 * This file currently provides implementations for the following cases:
 *   YCbCr => RGB color conversion only.
 *   Sampling ratios of 2h1v or 2h2v.
 *   No scaling needed at upsample time.
 *   Corner-aligned (non-CCIR601) sampling alignment.
 * Other special cases could be added, but in most applications these are
 * the only common cases. (For uncommon cases we fall back on the more
 * general code in jdsample.c and jdcolor.c.)
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

#ifndef UPSAMPLE_MERGING_SUPPORTED

/* Private subobject */

typedef struct {
  struct jpeg_upsampler pub; /* public fields */

  /* Pointer to routine to do actual upsampling/conversion of one row group */
  JMETHOD(void, upmethod, (j_decompress_ptr cinfo,
                          JSAMPIMAGE input_buf, JDIMENSION in_row_group_ctr,
                          JSAMPARRAY output_buf));

  /* Private state for YCC->RGB conversion */
  int * Cr_r_tab; /* => table for Cr to R conversion */
  int * Cb_b_tab; /* => table for Cb to B conversion */
  INT32 * Cr_g_tab; /* => table for Cr to G conversion */
  INT32 * Cb_g_tab; /* => table for Cb to G conversion */

  /* For 2:1 vertical sampling, we produce two output rows at a time.
   * We need a "spare" row buffer to hold the second output row if the
   * application provides just a one-row buffer; we also use the spare
   * to discard the dummy last row if the image height is odd.
   */
  JSAMPROW spare_row;
  boolean spare_full; /* T if spare buffer is occupied */

  JDIMENSION out_row_width; /* samples per output row */
  JDIMENSION rows_to_go; /* counts rows remaining in image */
} my_upsampler;

typedef my_upsampler * my_upsample_ptr;

#define SCALEBITS 16 /* speediest right-shift on some machines */
#define ONE_HALF ((INT32) 1 << (SCALEBITS-1))
#define FIX(x) ((INT32) ((x) * (1L<<SCALEBITS) + 0.5))

/*
 * Initialize tables for YCC->RGB colorspace conversion.
 * This is taken directly from jdcolor.c; see that file for more info.
 */

```

```

LOCAL(void)
build_ycc_rgb_table (j_decompress_ptr cinfo)
{
    my_upsample_ptr upsample = (my_upsample_ptr) cinfo->upsample;
    int i;
    INT32 x;
    SHIFT_TEMPS

    upsample->Cr_r_tab = (int *)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            (MAXJSAMPLE+1) * sizeof(int));
    upsample->Cb_b_tab = (int *)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            (MAXJSAMPLE+1) * sizeof(int));
    upsample->Cr_g_tab = (INT32 *)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            (MAXJSAMPLE+1) * sizeof(INT32));
    upsample->Cb_g_tab = (INT32 *)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            (MAXJSAMPLE+1) * sizeof(INT32));

    for (i = 0, x = -CENTERJSAMPLE; i <= MAXJSAMPLE; i++, x++) {
        /* i is the actual input pixel value, in the range 0..MAXJSAMPLE */
        /* The Cb or Cr value we are thinking of is x = i - CENTERJSAMPLE */
        /* Cr->R value is nearest int to 1.40200 * x */
        upsample->Cr_r_tab[i] = (int)
            RIGHT_SHIFT(FIX(1.40200) * x + ONE_HALF, SCALEBITS);
        /* Cb->B value is nearest int to 1.77200 * x */
        upsample->Cb_b_tab[i] = (int)
            RIGHT_SHIFT(FIX(1.77200) * x + ONE_HALF, SCALEBITS);
        /* Cr->G value is scaled-up -0.71414 * x */
        upsample->Cr_g_tab[i] = (- FIX(0.71414)) * x;
        /* Cb->G value is scaled-up -0.34414 * x */
        /* We also add in ONE_HALF so that need not do it in inner loop */
        upsample->Cb_g_tab[i] = (- FIX(0.34414)) * x + ONE_HALF;
    }

    /* Initialize for an upsampling pass.
    =
    */

METHODDEF(void)
start_pass_merged_upsample (j_decompress_ptr cinfo)
{
    my_upsample_ptr upsample = (my_upsample_ptr) cinfo->upsample;

    /* Mark the spare buffer empty */
    upsample->spare_full = FALSE;
    /* Initialize total-height counter for detecting bottom of image */
    upsample->rows_to_go = cinfo->output_height;
}

/*
 * Control routine to do upsampling (and color conversion).
 *
 * The control routine just handles the row buffering considerations.
 */

METHODDEF(void)
merged_2v_upsample (j_decompress_ptr cinfo,
    JSAMPIMAGE input_buf, JDIMENSION *in_row_group_ctr,
    JDIMENSION in_row_groups_avail,
    JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
    JDIMENSION out_rows_avail)
/* 2:1 vertical sampling case: may need a spare row. */
{
    my_upsample_ptr upsample = (my_upsample_ptr) cinfo->upsample;
    JSAMPROW work_ptrs[2];
    JDIMENSION num_rows; /* number of rows returned to caller */

    if (upsample->spare_full) {
        /* If we have a spare row saved from a previous cycle, just return it. */
        jcopy_sample_rows(& upsample->spare_row, 0, output_buf + *out_row_ctr, 0,
            1, upsample->out_row_width);
        num_rows = 1;
        upsample->spare_full = FALSE;
    }
}

```

```

    } else {
        /* Figure number of rows to return to caller. */
        num_rows = 2;
        /* Not more than the distance to the end of the image. */
        if (num_rows > upsample->rows_to_go)
            num_rows = upsample->rows_to_go;
        /* And not more than what the client can accept: */
        out_rows_avail -= *out_row_ctr;
        if (num_rows > out_rows_avail)
            num_rows = out_rows_avail;
        /* Create output pointer array for upsampler. */
        work_ptrs[0] = output_buf[*out_row_ctr];
        if (num_rows > 1) {
            work_ptrs[1] = output_buf[*out_row_ctr + 1];
        } else {
            work_ptrs[1] = upsample->spare_row;
            upsample->spare_full = TRUE;
        }
        /* Now do the upsampling. */
        (*upsample->upmethod) (cinfo, input_buf, *in_row_group_ctr, work_ptrs);
    }

    /* Adjust counts */
    *out_row_ctr += num_rows;
    upsample->rows_to_go -= num_rows;
    /* When the buffer is emptied, declare this input row group consumed */
    if (! upsample->spare_full)
        (*in_row_group_ctr)++;
}

METHODDEF(void)
merged_lv_upsample (j_decompress_ptr cinfo,
                    JSAMPIMAGE input_buf, JDIMENSION *in_row_group_ctr,
                    JDIMENSION in_row_groups_avail,
                    JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
                    JDIMENSION out_rows_avail)
/* 1:1 vertical sampling case: much easier, never need a spare row. */
{
    my_upsample_ptr upsample = (my_upsample_ptr) cinfo->upsample;

    /* Just do the upsampling. */
    (*upsample->upmethod) (cinfo, input_buf, *in_row_group_ctr,
                          output_buf + *out_row_ctr);
    /* Adjust counts */
    (*out_row_ctr)++;
    (*in_row_group_ctr)++;
}

/*
 * These are the routines invoked by the control routines to do
 * the actual upsampling/conversion. One row group is processed per call.
 *
 * Note: since we may be writing directly into application-supplied buffers,
 * we have to be honest about the output width; we can't assume the buffer
 * has been rounded up to an even width.
 */

/*
 * Upsample and color convert for the case of 2:1 horizontal and 1:1 vertical.
 */

METHODDEF(void)
h2v1_merged_upsample (j_decompress_ptr cinfo,
                      JSAMPIMAGE input_buf, JDIMENSION in_row_group_ctr,
                      JSAMPARRAY output_buf)
{
    my_upsample_ptr upsample = (my_upsample_ptr) cinfo->upsample;
    register int y, cred, cgreen, cbblue;
    int cb, cr;
    register JSAMPROW outptr;
    JSAMPROW inptr0, inptr1, inptr2;
    JDIMENSION col;
    /* copy these pointers into registers if possible */
    register JSAMPLE * range_limit = cinfo->sample_range_limit;
    int * Crctab = upsample->Cr_r_tab;
    int * Cbctab = upsample->Cb_b_tab;
    INT32 * Crgtab = upsample->Cr_g_tab;

```

```
INT32 * Cbgtab = upsample->Cb_g_tab;
SHIFT_TEMPS
```

```
inptr0 = input_buf[0][in_row_group_ctr];
inptr1 = input_buf[1][in_row_group_ctr];
inptr2 = input_buf[2][in_row_group_ctr];
outptr = output_buf[0];
/* Loop for each pair of output pixels */
for (col = cinfo->output_width >> 1; col > 0; col--) {
    /* Do the chroma part of the calculation */
    cb = GETJSAMPLE(*inptr1++);
    cr = GETJSAMPLE(*inptr2++);
    cred = Crrtab[cr];
    cgreen = (int) RIGHT_SHIFT(Cbgtab[cb] + Crgtab[cr], SCALEBITS);
    cbblue = Cbbtab[cb];
    /* Fetch 2 Y values and emit 2 pixels */
    y = GETJSAMPLE(*inptr0++);
    outptr[RGB_RED] = range_limit[y + cred];
    outptr[RGB_GREEN] = range_limit[y + cgreen];
    outptr[RGB_BLUE] = range_limit[y + cbblue];
    outptr += RGB_PIXELSIZE;
    y = GETJSAMPLE(*inptr0++);
    outptr[RGB_RED] = range_limit[y + cred];
    outptr[RGB_GREEN] = range_limit[y + cgreen];
    outptr[RGB_BLUE] = range_limit[y + cbblue];
    outptr += RGB_PIXELSIZE;
}
/* If image width is odd, do the last output column separately */
if (cinfo->output_width & 1) {
    cb = GETJSAMPLE(*inptr1);
    cr = GETJSAMPLE(*inptr2);
    cred = Crrtab[cr];
    cgreen = (int) RIGHT_SHIFT(Cbgtab[cb] + Crgtab[cr], SCALEBITS);
    cbblue = Cbbtab[cb];
    y = GETJSAMPLE(*inptr0);
    outptr[RGB_RED] = range_limit[y + cred];
    outptr[RGB_GREEN] = range_limit[y + cgreen];
    outptr[RGB_BLUE] = range_limit[y + cbblue];
}
```

\* Upsample and color convert for the case of 2:1 horizontal and 2:1 vertical.

```
METHODDEF(void)
```

```
h2v2_merged_upsample (j_decompress_ptr cinfo,
    JSAMPIMAGE input_buf, JDIMENSION in_row_group_ctr,
    JSAMPARRAY output_buf)
```

```
my_upsample_ptr upsample = (my_upsample_ptr) cinfo->upsample;
register int y, cred, cgreen, cbblue;
int cb, cr;
register JSAMPROW outptr0, outptr1;
JSAMPROW inptr00, inptr01, inptr1, inptr2;
JDIMENSION col;
/* copy these pointers into registers if possible */
register JSAMPLE * range_limit = cinfo->sample_range_limit;
int * Crrtab = upsample->Cr_r_tab;
int * Cbbtab = upsample->Cb_b_tab;
INT32 * Crgtab = upsample->Cr_g_tab;
INT32 * Cbgtab = upsample->Cb_g_tab;
SHIFT_TEMPS
```

```
inptr00 = input_buf[0][in_row_group_ctr*2];
inptr01 = input_buf[0][in_row_group_ctr*2 + 1];
inptr1 = input_buf[1][in_row_group_ctr];
inptr2 = input_buf[2][in_row_group_ctr];
outptr0 = output_buf[0];
outptr1 = output_buf[1];
/* Loop for each group of output pixels */
for (col = cinfo->output_width >> 1; col > 0; col--) {
    /* Do the chroma part of the calculation */
    cb = GETJSAMPLE(*inptr1++);
    cr = GETJSAMPLE(*inptr2++);
    cred = Crrtab[cr];
    cgreen = (int) RIGHT_SHIFT(Cbgtab[cb] + Crgtab[cr], SCALEBITS);
    cbblue = Cbbtab[cb];
    /* Fetch 4 Y values and emit 4 pixels */
```

```

    y = GETJSAMPLE(*inptr00++);
    outptr0[RGB_RED] = range_limit[y + cred];
    outptr0[RGB_GREEN] = range_limit[y + cgreen];
    outptr0[RGB_BLUE] = range_limit[y + cblue];
    outptr0 += RGB_PIXELSIZE;
    y = GETJSAMPLE(*inptr00++);
    outptr0[RGB_RED] = range_limit[y + cred];
    outptr0[RGB_GREEN] = range_limit[y + cgreen];
    outptr0[RGB_BLUE] = range_limit[y + cblue];
    outptr0 += RGB_PIXELSIZE;
    y = GETJSAMPLE(*inptr01++);
    outptr1[RGB_RED] = range_limit[y + cred];
    outptr1[RGB_GREEN] = range_limit[y + cgreen];
    outptr1[RGB_BLUE] = range_limit[y + cblue];
    outptr1 += RGB_PIXELSIZE;
    y = GETJSAMPLE(*inptr01++);
    outptr1[RGB_RED] = range_limit[y + cred];
    outptr1[RGB_GREEN] = range_limit[y + cgreen];
    outptr1[RGB_BLUE] = range_limit[y + cblue];
    outptr1 += RGB_PIXELSIZE;
}
/* If image width is odd, do the last output column separately */
if (cinfo->output_width & 1) {
    cb = GETJSAMPLE(*inptr1);
    cr = GETJSAMPLE(*inptr2);
    cred = Crrtab[cr];
    cgreen = (int) RIGHT_SHIFT(Cbgtab[cb] + Crgtab[cr], SCALEBITS);
    cblue = Cbbtab[cb];
    y = GETJSAMPLE(*inptr00);
    outptr0[RGB_RED] = range_limit[y + cred];
    outptr0[RGB_GREEN] = range_limit[y + cgreen];
    outptr0[RGB_BLUE] = range_limit[y + cblue];
    y = GETJSAMPLE(*inptr01);
    outptr1[RGB_RED] = range_limit[y + cred];
    outptr1[RGB_GREEN] = range_limit[y + cgreen];
    outptr1[RGB_BLUE] = range_limit[y + cblue];
}

/* Module initialization routine for merged upsampling/color conversion.
 * NB: this is called under the conditions determined by use_merged_upsample()
 * in jdmaster.c. That routine MUST correspond to the actual capabilities
 * of this module; no safety checks are made here.
 */

GLOBAL(void)
jinit_merged_upsampler (j_decompress_ptr cinfo)
{
    my_upsample_ptr upsample;

    upsample = (my_upsample_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            SIZEOF(my_upsampler));
    cinfo->upsample = (struct jpeg_upsampler *) upsample;
    upsample->pub.start_pass = start_pass_merged_upsample;
    upsample->pub.need_context_rows = FALSE;

    upsample->out_row_width = cinfo->output_width * cinfo->out_color_components;

    if (cinfo->max_v_samp_factor == 2) {
        upsample->pub.upsample = merged_2v_upsample;
        upsample->upmethod = h2v2_merged_upsample;
        /* Allocate a spare row buffer */
        upsample->spare_row = (JSAMPROW)
            (*cinfo->mem->alloc_large) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                (size_t) (upsample->out_row_width * SIZEOF(JSAMPLE)));
    } else {
        upsample->pub.upsample = merged_1v_upsample;
        upsample->upmethod = h2v1_merged_upsample;
        /* No spare row needed */
        upsample->spare_row = NULL;
    }

    build_ycc_rgb_table(cinfo);
}

#endif /* UPSAMPLE_MERGING_SUPPORTED */

```



```

/*
 * jdphuff.c
 *
 * Copyright (C) 1995-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains Huffman entropy decoding routines for progressive JPEG.
 *
 * Much of the complexity here has to do with supporting input suspension.
 * If the data source module demands suspension, we want to be able to back
 * up to the start of the current MCU. To do this, we copy state variables
 * into local working storage, and update them back to the permanent
 * storage only upon successful completion of an MCU.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jdphuff.h" /* Declarations shared with jdphuff.c */

#ifdef D_PROGRESSIVE_SUPPORTED

/*
 * Expanded entropy decoder object for progressive Huffman decoding.
 *
 * The savable_state subrecord contains fields that change within an MCU,
 * but must not be updated permanently until we complete the MCU.
 */

typedef struct {
  unsigned int EOBRUN; /* remaining EOBs in EOBRUN */
  int last_dc_val[MAX_COMPS_IN_SCAN]; /* last DC coef for each component */
} savable_state;

/* This macro is to work around compilers with missing or broken
 * structure assignment. You'll need to fix this code if you have
 * such a compiler and you change MAX_COMPS_IN_SCAN.
 */
#ifdef NO_STRUCT_ASSIGN
#define ASSIGN_STATE(dest,src) ((dest) = (src))
#else
#define ASSIGN_STATE(dest,src) \
  ((dest).EOBRUN = (src).EOBRUN, \
   (dest).last_dc_val[0] = (src).last_dc_val[0], \
   (dest).last_dc_val[1] = (src).last_dc_val[1], \
   (dest).last_dc_val[2] = (src).last_dc_val[2], \
   (dest).last_dc_val[3] = (src).last_dc_val[3])
#endif

#define NO_STRUCT_ASSIGN
#define ASSIGN_STATE(dest,src) ((dest) = (src))
#else
#define ASSIGN_STATE(dest,src) ((dest) = (src))
#endif

typedef struct {
  struct jpeg_entropy_decoder pub; /* public fields */

  /* These fields are loaded into local variables at start of each MCU.
   * In case of suspension, we exit WITHOUT updating them.
   */
  bitread_perm_state bitstate; /* Bit buffer at start of MCU */
  savable_state saved; /* Other state at start of MCU */

  /* These fields are NOT loaded into local working state. */
  unsigned int restarts_to_go; /* MCUs left in this restart interval */

  /* Pointers to derived tables (these workspaces have image lifespan) */
  d_derived_tbl * derived_tbls[NUM_HUFF_TBLS];

  d_derived_tbl * ac_derived_tbl; /* active table during an AC scan */
} phuff_entropy_decoder;

typedef phuff_entropy_decoder * phuff_entropy_ptr;

/* Forward declarations */
METHODDEF(boolean) decode_mcu_DC_first JPP((j_decompress_ptr cinfo,
                                           JBLOCKROW *MCU_data));
METHODDEF(boolean) decode_mcu_AC_first JPP((j_decompress_ptr cinfo,
                                           JBLOCKROW *MCU_data));

```

```

METHODDEF(boolean) decode_mcu_DC_refine JPP((j_decompress_ptr cinfo,
                                             JBLOCKROW *MCU_data));
METHODDEF(boolean) decode_mcu_AC_refine JPP((j_decompress_ptr cinfo,
                                             JBLOCKROW *MCU_data));

```

```

/*
 * Initialize for a Huffman-compressed scan.
 */

```

```

METHODDEF(void)
start_pass_phuff_decoder (j_decompress_ptr cinfo)
{
    phuff_entropy_ptr entropy = (phuff_entropy_ptr) cinfo->entropy;
    boolean is_DC_band, bad;
    int ci, coefi, tbl;
    int *coef_bit_ptr;
    jpeg_component_info * compptr;

    is_DC_band = (cinfo->Ss == 0);

    /* Validate scan parameters */
    bad = FALSE;
    if (is_DC_band) {
        if (cinfo->Se != 0)
            bad = TRUE;
    } else {
        /* need not check Ss/Se < 0 since they came from unsigned bytes */
        if (cinfo->Ss > cinfo->Se || cinfo->Se >= DCTSIZE2)
            bad = TRUE;
        /* AC scans may have only one component */
        if (cinfo->comps_in_scan != 1)
            bad = TRUE;
    }
    if (cinfo->Ah != 0) {
        /* Successive approximation refinement scan: must have A1 = Ah-1. */
        if (cinfo->A1 != cinfo->Ah-1)
            bad = TRUE;
    }
    if (cinfo->A1 > 13) /* need not check for < 0 */
        bad = TRUE;
    /* Arguably the maximum A1 value should be less than 13 for 8-bit precision,
     * but the spec doesn't say so, and we try to be liberal about what we
     * accept. Note: large A1 values could result in out-of-range DC
     * coefficients during early scans, leading to bizarre displays due to
     * overflows in the IDCT math. But we won't crash.
     */
    if (bad)
        ERREXIT4(cinfo, JERR_BAD_PROGRESSION,
                 cinfo->Ss, cinfo->Se, cinfo->Ah, cinfo->A1);
    /* Update progression status, and verify that scan order is legal.
     * Note that inter-scan inconsistencies are treated as warnings
     * not fatal errors ... not clear if this is right way to behave.
     */
    for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
        int cindex = cinfo->cur_comp_info[ci]->component_index;
        coef_bit_ptr = & cinfo->coef_bits[cindex][0];
        if (!is_DC_band && coef_bit_ptr[0] < 0) /* AC without prior DC scan */
            WARNMS2(cinfo, JWRN_BOGUS_PROGRESSION, cindex, 0);
        for (coefi = cinfo->Ss; coefi <= cinfo->Se; coefi++) {
            int expected = (coef_bit_ptr[coefi] < 0) ? 0 : coef_bit_ptr[coefi];
            if (cinfo->Ah != expected)
                WARNMS2(cinfo, JWRN_BOGUS_PROGRESSION, cindex, coefi);
            coef_bit_ptr[coefi] = cinfo->A1;
        }
    }

    /* Select MCU decoding routine */
    if (cinfo->Ah == 0) {
        if (is_DC_band)
            entropy->pub.decode_mcu = decode_mcu_DC_first;
        else
            entropy->pub.decode_mcu = decode_mcu_AC_first;
    } else {
        if (is_DC_band)
            entropy->pub.decode_mcu = decode_mcu_DC_refine;
        else
            entropy->pub.decode_mcu = decode_mcu_AC_refine;
    }
}

```

```

for (ci = 0; ci < cinfo->comps_in_scan; ci++) {
    compptr = cinfo->cur_comp_info[ci];
    /* Make sure requested tables are present, and compute derived tables.
     * We may build same derived table more than once, but it's not expensive.
     */
    if (is_DC_band) {
        if (cinfo->Ah == 0) { /* DC refinement needs no table */
            tbl = compptr->dc_tbl_no;
            jpeg_make_d_derived_tbl(cinfo, TRUE, tbl,
                                   & entropy->derived_tbls[tbl]);
        }
        else {
            tbl = compptr->ac_tbl_no;
            jpeg_make_d_derived_tbl(cinfo, FALSE, tbl,
                                   & entropy->derived_tbls[tbl]);
            /* remember the single active table */
            entropy->ac_derived_tbl = entropy->derived_tbls[tbl];
        }
        /* Initialize DC predictions to 0 */
        entropy->saved.last_dc_val[ci] = 0;
    }

    /* Initialize bitread state variables */
    entropy->bitstate.bits_left = 0;
    entropy->bitstate.get_buffer = 0; /* unnecessary, but keeps Purify quiet */
    entropy->pub.insufficient_data = FALSE;

    /* Initialize private state variables */
    entropy->saved.EOBRUN = 0;

    /* Initialize restart counter */
    entropy->restarts_to_go = cinfo->restart_interval;
}

/*
 * Figure F.12: extend sign bit.
 * On some machines, a shift and add will be faster than a table lookup.
 */
#ifdef AVOID_TABLES
#define HUFF_EXTEND(x,s) ((x) < (1<<((s)-1)) ? (x) + (((-1)<<(s)) + 1) : (x))
#else
#define HUFF_EXTEND(x,s) ((x) < extend_test[s] ? (x) + extend_offset[s] : (x))
static const int extend_test[16] = /* entry n is 2**(n-1) */
{ 0, 0x0001, 0x0002, 0x0004, 0x0008, 0x0010, 0x0020, 0x0040, 0x0080,
  0x0100, 0x0200, 0x0400, 0x0800, 0x1000, 0x2000, 0x4000 };
static const int extend_offset[16] = /* entry n is (-1 << n) + 1 */
{ 0, ((-1)<<1) + 1, ((-1)<<2) + 1, ((-1)<<3) + 1, ((-1)<<4) + 1,
  ((-1)<<5) + 1, ((-1)<<6) + 1, ((-1)<<7) + 1, ((-1)<<8) + 1,
  ((-1)<<9) + 1, ((-1)<<10) + 1, ((-1)<<11) + 1, ((-1)<<12) + 1,
  ((-1)<<13) + 1, ((-1)<<14) + 1, ((-1)<<15) + 1 };
#endif /* AVOID_TABLES */

/*
 * Check for a restart marker & resynchronize decoder.
 * Returns FALSE if must suspend.
 */
LOCAL(boolean)
process_restart (j_decompress_ptr cinfo)
{
    phuff_entropy_ptr entropy = (phuff_entropy_ptr) cinfo->entropy;
    int ci;

    /* Throw away any unused bits remaining in bit buffer; */
    /* include any full bytes in next_marker's count of discarded bytes */
    cinfo->marker->discarded_bytes += entropy->bitstate.bits_left / 8;
    entropy->bitstate.bits_left = 0;

    /* Advance past the RSTn marker */
    if (! (*cinfo->marker->read_restart_marker) (cinfo))
        return FALSE;
}

```

```

/* Re-initialize DC predictions to 0 */
for (ci = 0; ci < cinfo->comps_in_scan; ci++)
    entropy->saved.last_dc_val[ci] = 0;
/* Re-init EOB run count, too */
entropy->saved.EOBRUN = 0;

/* Reset restart counter */
entropy->restarts_to_go = cinfo->restart_interval;

/* Reset out-of-data flag, unless read_restart_marker left us smack up
 * against a marker. In that case we will end up treating the next data
 * segment as empty, and we can avoid producing bogus output pixels by
 * leaving the flag set.
 */
if (cinfo->unread_marker == 0)
    entropy->pub.insufficient_data = FALSE;

return TRUE;
}

/*
 * Huffman MCU decoding.
 * Each of these routines decodes and returns one MCU's worth of
 * Huffman-compressed coefficients.
 * The coefficients are reordered from zigzag order into natural array order,
 * but are not dequantized.
 *
 * The i'th block of the MCU is stored into the block pointed to by
 * MCU_data[i]. WE ASSUME THIS AREA IS INITIALLY ZEROED BY THE CALLER.
 *
 * We return FALSE if data source requested suspension. In that case no
 * changes have been made to permanent state. (Exception: some output
 * coefficients may already have been assigned. This is harmless for
 * spectral selection, since we'll just re-assign them on the next call.
 * Successive approximation AC refinement has to be more careful, however.)
 */

/* MCU decoding for DC initial scan (either spectral selection,
 * or first pass of successive approximation).
 */
=
METHODDEF(boolean)
decode_mcu_DC_first (j_decompress_ptr cinfo, JBLOCKROW *MCU_data)
{
    phuff_entropy_ptr entropy = (phuff_entropy_ptr) cinfo->entropy;
    int Al = cinfo->Al;
    register int s, r;
    int blk, ci;
    JBLOCKROW block;
    BITREAD_STATE_VARS;
    savable_state state;
    d_derived_tbl *tbl;
    jpeg_component_info *comp_ptr;

    /* Process restart marker if needed; may have to suspend */
    if (cinfo->restart_interval) {
        if (entropy->restarts_to_go == 0)
            if (! process_restart(cinfo))
                return FALSE;
    }

    /* If we've run out of data, just leave the MCU set to zeroes.
     * This way, we return uniform gray for the remainder of the segment.
     */
    if (! entropy->pub.insufficient_data) {

        /* Load up working state */
        BITREAD_LOAD_STATE(cinfo, entropy->bitstate);
        ASSIGN_STATE(state, entropy->saved);

        /* Outer loop handles each block in the MCU */

        for (blk = 0; blk < cinfo->blocks_in_MCU; blk++) {
            block = MCU_data[blk];
            ci = cinfo->MCU_membership[blk];
            comp_ptr = cinfo->cur_comp_info[ci];
            tbl = entropy->derived_tbls[comp_ptr->dc_tbl_no];

```

```

/* Decode a single block's worth of coefficients */

/* Section F.2.2.1: decode the DC coefficient difference */
HUFF_DECODE(s, br_state, tbl, return FALSE, label1);
if (s) {
    CHECK_BIT_BUFFER(br_state, s, return FALSE);
    r = GET_BITS(s);
    s = HUFF_EXTEND(r, s);
}

/* Convert DC difference to actual value, update last_dc_val */
s += state.last_dc_val[ci];
state.last_dc_val[ci] = s;
/* Scale and output the coefficient (assumes jpeg_natural_order[0]=0) */
(*block)[0] = (JCOEF) (s << A1);
}

/* Completed MCU, so update state */
BITREAD_SAVE_STATE(cinfo, entropy->bitstate);
ASSIGN_STATE(entropy->saved, state);
}

/* Account for restart interval (no-op if not using restarts) */
entropy->restarts_to_go--;

return TRUE;
}

/*
 * MCU decoding for AC initial scan (either spectral selection,
 * or first pass of successive approximation).
 */
METHODDEF(boolean)
decode_mcu_AC_first (j_decompress_ptr cinfo, JBLOCKROW *MCU_data)
{
    phuff_entropy_ptr entropy = (phuff_entropy_ptr) cinfo->entropy;
    int Se = cinfo->Se;
    int A1 = cinfo->A1;
    register int s, k, r;
    unsigned int EOBRUN;
    JBLOCKROW block;
    BITREAD_STATE_VARS;
    d_derived_tbl * tbl;

    /* Process restart marker if needed; may have to suspend */
    if (cinfo->restart_interval) {
        if (entropy->restarts_to_go == 0)
            if (! process_restart(cinfo))
                return FALSE;
    }

    /* If we've run out of data, just leave the MCU set to zeroes.
     * This way, we return uniform gray for the remainder of the segment.
     */
    if (! entropy->pub.insufficient_data) {
        /* Load up working state.
         * We can avoid loading/saving bitread state if in an EOB run.
         */
        EOBRUN = entropy->saved.EOBRUN; /* only part of saved state we need */

        /* There is always only one block per MCU */

        if (EOBRUN > 0) /* if it's a band of zeroes... */
            EOBRUN--; /* ...process it now (we do nothing) */
        else {
            BITREAD_LOAD_STATE(cinfo, entropy->bitstate);
            block = MCU_data[0];
            tbl = entropy->ac_derived_tbl;

            for (k = cinfo->Ss; k <= Se; k++) {
                HUFF_DECODE(s, br_state, tbl, return FALSE, label2);
                r = s >> 4;
                s &= 15;
                if (s) {
                    k += r;
                    CHECK_BIT_BUFFER(br_state, s, return FALSE);

```

```

    r = GET_BITS(s);
    s = HUFF_EXTEND(r, s);
    /* Scale and output coefficient in natural (dezigzagged) order */
    (*block)[jpeg_natural_order[k]] = (JCOEF) (s << A1);
} else {
    if (r == 15) { /* ZRL */
        k += 15; /* skip 15 zeroes in band */
    } else { /* EOB, run length is 2^r + appended bits */
        EOBRUN = 1 << r;
        if (r) { /* EOB, r > 0 */
            CHECK_BIT_BUFFER(br_state, r, return FALSE);
            r = GET_BITS(r);
            EOBRUN += r;
        }
        EOBRUN--; /* this band is processed at this moment */
        break; /* force end-of-band */
    }
}

BITREAD_SAVE_STATE(cinfo, entropy->bitstate);
}

/* Completed MCU, so update state */
entropy->saved.EOBRUN = EOBRUN; /* only part of saved state we need */
}

/* Account for restart interval (no-op if not using restarts) */
entropy->restarts_to_go--;

return TRUE;
}

/*
 * MCU decoding for DC successive approximation refinement scan.
 * Note: we assume such scans can be multi-component, although the spec
 * is not very clear on the point.
 */
METHODDEF(boolean)
decode_mcu_DC_refine (j_decompress_ptr cinfo, JBLOCKROW *MCU_data)
{
    phuff_entropy_ptr entropy = (phuff_entropy_ptr) cinfo->entropy;
    int p1 = 1 << cinfo->A1; /* 1 in the bit position being coded */
    int blkn;
    JBLOCKROW block;
    BITREAD_STATE_VARS;

    /* Process restart marker if needed; may have to suspend */
    if (cinfo->restart_interval) {
        if (entropy->restarts_to_go == 0)
            if (! process_restart(cinfo))
                return FALSE;
    }

    /* Not worth the cycles to check insufficient_data here,
     * since we will not change the data anyway if we read zeroes.
     */

    /* Load up working state */
    BITREAD_LOAD_STATE(cinfo, entropy->bitstate);

    /* Outer loop handles each block in the MCU */
    for (blkn = 0; blkn < cinfo->blocks_in_MCU; blkn++) {
        block = MCU_data[blkn];

        /* Encoded data is simply the next bit of the two's-complement DC value */
        CHECK_BIT_BUFFER(br_state, 1, return FALSE);
        if (GET_BITS(1))
            (*block)[0] |= p1;
        /* Note: since we use |=, repeating the assignment later is safe */
    }

    /* Completed MCU, so update state */
    BITREAD_SAVE_STATE(cinfo, entropy->bitstate);

    /* Account for restart interval (no-op if not using restarts) */
    entropy->restarts_to_go--;
}

```

```

return TRUE;
}

/*
 * MCU decoding for AC successive approximation refinement scan.
 */

METHODDEF(boolean)
decode_mcu_AC_refine (j_decompress_ptr cinfo, JBLOCKROW *MCU_data)
{
    phuff_entropy_ptr entropy = (phuff_entropy_ptr) cinfo->entropy;
    int Se = cinfo->Se;
    int p1 = 1 << cinfo->A1; /* 1 in the bit position being coded */
    int m1 = (-1) << cinfo->A1; /* -1 in the bit position being coded */
    register int s, k, r;
    unsigned int EOBRUN;
    JBLOCKROW block;
    JCOEFPTR thiscoef;
    BITREAD_STATE_VARS;
    d_derived_tbl * tbl;
    int num_newnz;
    int newnz_pos[DCTSIZE2];

    /* Process restart marker if needed; may have to suspend */
    if (cinfo->restart_interval) {
        if (entropy->restarts_to_go == 0)
            if (! process_restart(cinfo))
                return FALSE;
    }

    /* If we've run out of data, don't modify the MCU.
     */
    if (! entropy->pub.insufficient_data) {
        /* Load up working state */
        BITREAD_LOAD_STATE(cinfo, entropy->bitstate);
        EOBRUN = entropy->saved.EOBRUN; /* only part of saved state we need */

        /* There is always only one block per MCU */
        block = MCU_data[0];
        tbl = entropy->ac_derived_tbl;

        /* If we are forced to suspend, we must undo the assignments to any newly
         * nonzero coefficients in the block, because otherwise we'd get confused
         * next time about which coefficients were already nonzero.
         * But we need not undo addition of bits to already-nonzero coefficients;
         * instead, we can test the current bit to see if we already did it.
         */
        num_newnz = 0;

        /* initialize coefficient loop counter to start of band */
        k = cinfo->Ss;

        if (EOBRUN == 0) {
            for (; k <= Se; k++) {
                HUFF_DECODE(s, br_state, tbl, goto undoit, label3);
                r = s >> 4;
                s &= 15;
                if (s) {
                    if (s != 1) /* size of new coef should always be 1 */
                        WARNMS(cinfo, JWRN_HUFF_BAD_CODE);
                    CHECK_BIT_BUFFER(br_state, 1, goto undoit);
                    if (GET_BITS(1))
                        s = p1; /* newly nonzero coef is positive */
                    else
                        s = m1; /* newly nonzero coef is negative */
                } else {
                    if (r != 15) {
                        EOBRUN = 1 << r; /* EOBr, run length is 2^r + appended bits */
                        if (r) {
                            CHECK_BIT_BUFFER(br_state, r, goto undoit);
                            r = GET_BITS(r);
                            EOBRUN += r;
                        }
                        break; /* rest of block is handled by EOB logic */
                    }
                }
                /* note s = 0 for processing ZRL */
            }
        }
    }
}

```

```

/* Advance over already-nonzero coefs and r still-zero coefs,
 * appending correction bits to the nonzeros. A correction bit is 1
 * if the absolute value of the coefficient must be increased.
 */
do {
    thiscoef = *block + jpeg_natural_order[k];
    if (*thiscoef != 0) {
        CHECK_BIT_BUFFER(br_state, 1, goto undoit);
        if (GET_BITS(1)) {
            if ((*thiscoef & p1) == 0) { /* do nothing if already set it */
                if (*thiscoef >= 0)
                    *thiscoef += p1;
                else
                    *thiscoef += m1;
            }
        } else {
            if (--r < 0)
                break; /* reached target zero coefficient */
        }
        k++;
    } while (k <= Se);
    if (s) {
        int pos = jpeg_natural_order[k];
        /* Output newly nonzero coefficient */
        (*block)[pos] = (JCOEF) s;
        /* Remember its position in case we have to suspend */
        newnz_pos[num_newnz++] = pos;
    }
}

if (EOBRUN > 0) {
    /* Scan any remaining coefficient positions after the end-of-band
     * (the last newly nonzero coefficient, if any). Append a correction
     * bit to each already-nonzero coefficient. A correction bit is 1
     * if the absolute value of the coefficient must be increased.
     */
    for (; k <= Se; k++) {
        thiscoef = *block + jpeg_natural_order[k];
        if (*thiscoef != 0) {
            CHECK_BIT_BUFFER(br_state, 1, goto undoit);
            if (GET_BITS(1)) {
                if ((*thiscoef & p1) == 0) { /* do nothing if already changed it */
                    if (*thiscoef >= 0)
                        *thiscoef += p1;
                    else
                        *thiscoef += m1;
                }
            }
        }
    }
    /* Count one block completed in EOB run */
    EOBRUN--;
}

/* Completed MCU, so update state */
BITREAD_SAVE_STATE(cinfo, entropy->bitstate);
entropy->saved.EOBRUN = EOBRUN; /* only part of saved state we need */
}

/* Account for restart interval (no-op if not using restarts) */
entropy->restarts_to_go--;

return TRUE;

undoit:
/* Re-zero any output coefficients that we made newly nonzero */
while (num_newnz > 0)
    (*block)[newnz_pos[--num_newnz]] = 0;

return FALSE;
}

/*
 * Module initialization routine for progressive Huffman entropy decoding.
 */
GLOBAL(void)

```



```

jinit_phuff_decoder (j_decompress_ptr cinfo)
{
    phuff_entropy_ptr entropy;
    int *coef_bit_ptr;
    int ci, i;

    entropy = (phuff_entropy_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            SIZEOF(phuff_entropy_decoder));
    cinfo->entropy = (struct jpeg_entropy_decoder *) entropy;
    entropy->pub.start_pass = start_pass_phuff_decoder;

    /* Mark derived tables unallocated */
    for (i = 0; i < NUM_HUFF_TBLS; i++) {
        entropy->derived_tbls[i] = NULL;
    }

    /* Create progression status table */
    cinfo->coef_bits = (int (*)[DCTSIZE2])
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            cinfo->num_components*DCTSIZE2*SIZEOF(int));
    coef_bit_ptr = & cinfo->coef_bits[0][0];
    for (ci = 0; ci < cinfo->num_components; ci++)
        for (i = 0; i < DCTSIZE2; i++)
            *coef_bit_ptr++ = -1;
}

#endif /* D_PROGRESSIVE_SUPPORTED */

```

```

/*
 * jdpostct.c
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains the decompression postprocessing controller.
 * This controller manages the upsampling, color conversion, and color
 * quantization/reduction steps; specifically, it controls the buffering
 * between upsample/color conversion and color quantization/reduction.
 *
 * If no color quantization/reduction is required, then this module has no
 * work to do, and it just hands off to the upsample/color conversion code.
 * An integrated upsample/convert/quantize process would replace this module
 * entirely.
 */

```

```

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

```

```

/* Private buffer controller object */

```

```

typedef struct {
  struct jpeg_d_post_controller pub; /* public fields */

  /* Color quantization source buffer: this holds output data from
   * the upsample/color conversion step to be passed to the quantizer.
   * For two-pass color quantization, we need a full-image buffer;
   * for one-pass operation, a strip buffer is sufficient.
   */
  JSAMPARRAY whole_image; /* virtual array, or NULL if one-pass */
  JSAMPARRAY buffer; /* strip buffer, or current strip of virtual */
  JDIMENSION strip_height; /* buffer size in rows */
  /* for two-pass mode only: */
  JDIMENSION starting_row; /* row # of first row in current strip */
  JDIMENSION next_row; /* index of next row to fill/empty in strip */
} my_post_controller;

typedef my_post_controller * my_post_ptr;

```

```

/* Forward declarations */
METHODDEF(void) post_process_1pass
  JPP((j_decompress_ptr cinfo,
       JSAMPIMAGE input_buf, JDIMENSION *in_row_group_ctr,
       JDIMENSION in_row_groups_avail,
       JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
       JDIMENSION out_rows_avail));
#ifdef QUANT_2PASS_SUPPORTED
METHODDEF(void) post_process_prepass
  JPP((j_decompress_ptr cinfo,
       JSAMPIMAGE input_buf, JDIMENSION *in_row_group_ctr,
       JDIMENSION in_row_groups_avail,
       JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
       JDIMENSION out_rows_avail));
METHODDEF(void) post_process_2pass
  JPP((j_decompress_ptr cinfo,
       JSAMPIMAGE input_buf, JDIMENSION *in_row_group_ctr,
       JDIMENSION in_row_groups_avail,
       JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
       JDIMENSION out_rows_avail));
#endif
#endif

```

```

/*
 * Initialize for a processing pass.
 */
METHODDEF(void)
start_pass_dpost (j_decompress_ptr cinfo, J_BUF_MODE pass_mode)
{
  my_post_ptr post = (my_post_ptr) cinfo->post;

  switch (pass_mode) {
    case JBUF_PASS_THRU:
      if (cinfo->quantize_colors) {
        /* Single-pass processing with color quantization. */

```

```

    post->pub.post_process_data = post_process_lpass;
    /* We could be doing buffered-image output before starting a 2-pass
     * color quantization; in that case, jinit_d_post_controller did not
     * allocate a strip buffer. Use the virtual-array buffer as workspace.
     */
    if (post->buffer == NULL) {
        post->buffer = (*cinfo->mem->access_virt_sarray)
            ((j_common_ptr) cinfo, post->whole_image,
             (JDIMENSION) 0, post->strip_height, TRUE);
    }
} else {
    /* For single-pass processing without color quantization,
     * I have no work to do; just call the upsampler directly.
     */
    post->pub.post_process_data = cinfo->upsample->upsample;
}
break;
#endif QUANT_2PASS_SUPPORTED
case JBUF_SAVE_AND_PASS:
    /* First pass of 2-pass quantization */
    if (post->whole_image == NULL)
        ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
    post->pub.post_process_data = post_process_prepass;
    break;
case JBUF_CRANK_DEST:
    /* Second pass of 2-pass quantization */
    if (post->whole_image == NULL)
        ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
    post->pub.post_process_data = post_process_2pass;
    break;
#endif /* QUANT_2PASS_SUPPORTED */
default:
    ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
    break;
}
post->starting_row = post->next_row = 0;

/* Process some data in the one-pass (strip buffer) case.
 * This is used for color precision reduction as well as one-pass quantization.
 */
METHODDEF(void)
post_process_lpass (j_decompress_ptr cinfo,
                    JSAMPIMAGE input_buf, JDIMENSION *in_row_group_ctr,
                    JDIMENSION in_row_groups_avail,
                    JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
                    JDIMENSION out_rows_avail)
{
    my_post_ptr post = (my_post_ptr) cinfo->post;
    JDIMENSION num_rows, max_rows;

    /* Fill the buffer, but not more than what we can dump out in one go. */
    /* Note we rely on the upsampler to detect bottom of image. */
    max_rows = out_rows_avail - *out_row_ctr;
    if (max_rows > post->strip_height)
        max_rows = post->strip_height;
    num_rows = 0;
    (*cinfo->upsample->upsample) (cinfo,
                                input_buf, in_row_group_ctr, in_row_groups_avail,
                                post->buffer, &num_rows, max_rows);
    /* Quantize and emit data. */
    (*cinfo->cquantize->color_quantize) (cinfo,
                                         post->buffer, output_buf + *out_row_ctr, (int) num_rows);
    *out_row_ctr += num_rows;
}

#ifdef QUANT_2PASS_SUPPORTED
/*
 * Process some data in the first pass of 2-pass quantization.
 */
METHODDEF(void)
post_process_prepass (j_decompress_ptr cinfo,
                     JSAMPIMAGE input_buf, JDIMENSION *in_row_group_ctr,
                     JDIMENSION in_row_groups_avail,

```

```

        JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
        JDIMENSION out_rows_avail)
{
    my_post_ptr post = (my_post_ptr) cinfo->post;
    JDIMENSION old_next_row, num_rows;

    /* Reposition virtual buffer if at start of strip. */
    if (post->next_row == 0) {
        post->buffer = (*cinfo->mem->access_virt_sarray)
            ((j_common_ptr) cinfo, post->whole_image,
            post->starting_row, post->strip_height, TRUE);
    }

    /* Upsample some data (up to a strip height's worth). */
    old_next_row = post->next_row;
    (*cinfo->upsample->upsample) (cinfo,
        input_buf, in_row_group_ctr, in_row_groups_avail,
        post->buffer, &post->next_row, post->strip_height);

    /* Allow quantizer to scan new data. No data is emitted, */
    /* but we advance out_row_ctr so outer loop can tell when we're done. */
    if (post->next_row > old_next_row) {
        num_rows = post->next_row - old_next_row;
        (*cinfo->cquantize->color_quantize) (cinfo, post->buffer + old_next_row,
            (JSAMPARRAY) NULL, (int) num_rows);
        *out_row_ctr += num_rows;
    }

    /* Advance if we filled the strip. */
    if (post->next_row >= post->strip_height) {
        post->starting_row += post->strip_height;
        post->next_row = 0;
    }

    /* Process some data in the second pass of 2-pass quantization. */
}

METHODDEF(void)
post_process_2pass (j_decompress_ptr cinfo,
    JSAMPIMAGE input_buf, JDIMENSION *in_row_group_ctr,
    JDIMENSION in_row_groups_avail,
    JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
    JDIMENSION out_rows_avail)
{
    my_post_ptr post = (my_post_ptr) cinfo->post;
    JDIMENSION num_rows, max_rows;

    /* Reposition virtual buffer if at start of strip. */
    if (post->next_row == 0) {
        post->buffer = (*cinfo->mem->access_virt_sarray)
            ((j_common_ptr) cinfo, post->whole_image,
            post->starting_row, post->strip_height, FALSE);
    }

    /* Determine number of rows to emit. */
    num_rows = post->strip_height - post->next_row; /* available in strip */
    max_rows = out_rows_avail - *out_row_ctr; /* available in output area */
    if (num_rows > max_rows)
        num_rows = max_rows;
    /* We have to check bottom of image here, can't depend on upsampler. */
    max_rows = cinfo->output_height - post->starting_row;
    if (num_rows > max_rows)
        num_rows = max_rows;

    /* Quantize and emit data. */
    (*cinfo->cquantize->color_quantize) (cinfo,
        post->buffer + post->next_row, output_buf + *out_row_ctr,
        (int) num_rows);
    *out_row_ctr += num_rows;

    /* Advance if we filled the strip. */
    post->next_row += num_rows;
    if (post->next_row >= post->strip_height) {
        post->starting_row += post->strip_height;
        post->next_row = 0;
    }
}

```

```

#endif /* QUANT_2PASS_SUPPORTED */

/*
 * Initialize postprocessing controller.
 */

GLOBAL(void)
jinit_d_post_controller (j_decompress_ptr cinfo, boolean need_full_buffer)
{
    my_post_ptr post;

    post = (my_post_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            sizeof(my_post_controller));
    cinfo->post = (struct jpeg_d_post_controller *) post;
    post->pub.start_pass = start_pass_dpost;
    post->whole_image = NULL; /* flag for no virtual arrays */
    post->buffer = NULL;      /* flag for no strip buffer */

    /* Create the quantization buffer, if needed */
    if (cinfo->quantize_colors) {
        /* The buffer strip height is max_v_samp_factor, which is typically
         * an efficient number of rows for upsampling to return.
         * (In the presence of output rescaling, we might want to be smarter?)
         */
        post->strip_height = (JDIMENSION) cinfo->max_v_samp_factor;
        if (need_full_buffer) {
            /* Two-pass color quantization: need full-image storage. */
            /* We round up the number of rows to a multiple of the strip height. */
#ifdef QUANT_2PASS_SUPPORTED
            post->whole_image = (*cinfo->mem->request_virt_sarray)
                ((j_common_ptr) cinfo, JPOOL_IMAGE, FALSE,
                    cinfo->output_width * cinfo->out_color_components,
                    (JDIMENSION) jround_up((long) cinfo->output_height,
                        (long) post->strip_height),
                    post->strip_height);
#else
            ERREXIT(cinfo, JERR_BAD_BUFFER_MODE);
#endif
        }
    }
    /* QUANT_2PASS_SUPPORTED */
} else {
    /* One-pass color quantization: just make a strip buffer. */
    post->buffer = (*cinfo->mem->alloc_sarray)
        ((j_common_ptr) cinfo, JPOOL_IMAGE,
            cinfo->output_width * cinfo->out_color_components,
            post->strip_height);
}
}

```

```

/*
 * jdsample.c
 *
 * Copyright (C) 1991-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains upsampling routines.
 *
 * Upsampling input data is counted in "row groups". A row group
 * is defined to be (v_samp_factor * DCT_scaled_size / min_DCT_scaled_size)
 * sample rows of each component. Upsampling will normally produce
 * max_v_samp_factor pixel rows from each row group (but this could vary
 * if the upsampler is applying a scale factor of its own).
 *
 * An excellent reference for image resampling is
 * Digital Image Warping, George Wolberg, 1990.
 * Pub. by IEEE Computer Society Press, Los Alamitos, CA. ISBN 0-8186-8944-7.
 */

```

```

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

```

```

/* Pointer to routine to upsample a single component */
typedef JMETHOD(void, upsample1_ptr,
  (j_decompress_ptr cinfo, jpeg_component_info * comp_ptr,
   JSAMPARRAY input_data, JSAMPARRAY * output_data_ptr));

```

```

/* Private subobject */

```

```

typedef struct {
  struct jpeg_upsampler pub; /* public fields */

  /* Color conversion buffer. When using separate upsampling and color
   * conversion steps, this buffer holds one upsampled row group until it
   * has been color converted and output.
   * Note: we do not allocate any storage for component(s) which are full-size,
   * ie do not need rescaling. The corresponding entry of color_buf[] is
   * simply set to point to the input data array, thereby avoiding copying.
   */
  JSAMPARRAY color_buf[MAX_COMPONENTS];

  /* Per-component upsampling method pointers */
  upsample1_ptr methods[MAX_COMPONENTS];

  int next_row_out; /* counts rows emitted from color_buf */
  JDIMENSION rows_to_go; /* counts rows remaining in image */

  /* Height of an input row group for each component. */
  int rowgroup_height[MAX_COMPONENTS];

  /* These arrays save pixel expansion factors so that int_expand need not
   * recompute them each time. They are unused for other upsampling methods.
   */
  UINT8 h_expand[MAX_COMPONENTS];
  UINT8 v_expand[MAX_COMPONENTS];
} my_upsampler;

```

```

typedef my_upsampler * my_upsample_ptr;

```

```

/*
 * Initialize for an upsampling pass.
 */

```

```

METHODDEF(void)
start_pass_upsample (j_decompress_ptr cinfo)
{
  my_upsample_ptr upsample = (my_upsample_ptr) cinfo->upsample;

  /* Mark the conversion buffer empty */
  upsample->next_row_out = cinfo->max_v_samp_factor;
  /* Initialize total-height counter for detecting bottom of image */
  upsample->rows_to_go = cinfo->output_height;
}

```

```

/*

```

```

* Control routine to do upsampling (and color conversion).
*
* In this version we upsample each component independently.
* We upsample one row group into the conversion buffer, then apply
* color conversion a row at a time.
*/

```

```

METHODDEF(void)

```

```

sep_upsample (j_decompress_ptr cinfo,
              JSAMPIMAGE input_buf, JDIMENSION *in_row_group_ctr,
              JDIMENSION in_row_groups_avail,
              JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
              JDIMENSION out_rows_avail)

```

```

{
    my_upsample_ptr upsample = (my_upsample_ptr) cinfo->upsample;
    int ci;
    jpeg_component_info * compptr;
    JDIMENSION num_rows;

    /* Fill the conversion buffer, if it's empty */
    if (upsample->next_row_out >= cinfo->max_v_samp_factor) {
        for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
             ci++, compptr++) {
            /* Invoke per-component upsample method. Notice we pass a POINTER
             * to color_buf[ci], so that fullsize_upsample can change it.
             */
            (*upsample->methods[ci]) (cinfo, compptr,
                                     input_buf[ci] + (*in_row_group_ctr * upsample->rowgroup_height[ci]),
                                     upsample->color_buf + ci);
        }
        upsample->next_row_out = 0;

        /* Color-convert and emit rows */

        /* How many we have in the buffer: */
        num_rows = (JDIMENSION) (cinfo->max_v_samp_factor - upsample->next_row_out);
        /* Not more than the distance to the end of the image. Need this test
        * in case the image height is not a multiple of max_v_samp_factor:
        */
        if (num_rows > upsample->rows_to_go)
            num_rows = upsample->rows_to_go;
        /* And not more than what the client can accept: */
        out_rows_avail -= *out_row_ctr;
        if (num_rows > out_rows_avail)
            num_rows = out_rows_avail;

        (*cinfo->cconvert->color_convert) (cinfo, upsample->color_buf,
                                         (JDIMENSION) upsample->next_row_out,
                                         output_buf + *out_row_ctr,
                                         (int) num_rows);

        /* Adjust counts */
        *out_row_ctr += num_rows;
        upsample->rows_to_go -= num_rows;
        upsample->next_row_out += num_rows;
        /* When the buffer is emptied, declare this input row group consumed */
        if (upsample->next_row_out >= cinfo->max_v_samp_factor)
            (*in_row_group_ctr)++;
    }
}

```

```

/*
 * These are the routines invoked by sep_upsample to upsample pixel values
 * of a single component. One row group is processed per call.
 */

```

```

/*
 * For full-size components, we just make color_buf[ci] point at the
 * input buffer, and thus avoid copying any data. Note that this is
 * safe only because sep_upsample doesn't declare the input row group
 * "consumed" until we are done color converting and emitting it.
 */

```

```

METHODDEF(void)

```

```

fullsize_upsample (j_decompress_ptr cinfo, jpeg_component_info * compptr,
                  JSAMPARRAY input_data, JSAMPARRAY * output_data_ptr)

```

```

{
    *output_data_ptr = input_data;
}

```

```

}

/*
 * This is a no-op version used for "uninteresting" components.
 * These components will not be referenced by color conversion.
 */

METHODDEF(void)
noop_upsample (j_decompress_ptr cinfo, jpeg_component_info * comptr,
               JSAMPARRAY input_data, JSAMPARRAY * output_data_ptr)
{
    *output_data_ptr = NULL; /* safety check */
}

/*
 * This version handles any integral sampling ratios.
 * This is not used for typical JPEG files, so it need not be fast.
 * Nor, for that matter, is it particularly accurate: the algorithm is
 * simple replication of the input pixel onto the corresponding output
 * pixels. The hi-falutin sampling literature refers to this as a
 * "box filter". A box filter tends to introduce visible artifacts,
 * so if you are actually going to use 3:1 or 4:1 sampling ratios
 * you would be well advised to improve this code.
 */

METHODDEF(void)
int_upsample (j_decompress_ptr cinfo, jpeg_component_info * comptr,
              JSAMPARRAY input_data, JSAMPARRAY * output_data_ptr)
{
    my_upsample_ptr upsample = (my_upsample_ptr) cinfo->upsample;
    JSAMPARRAY output_data = *output_data_ptr;
    register JSAMPROW inptr, outptr;
    register JSAMPLE invalue;
    register int h;
    JSAMPROW outend;
    int h_expand, v_expand;
    int inrow, outrow;

    h_expand = upsample->h_expand[comptr->component_index];
    v_expand = upsample->v_expand[comptr->component_index];

    inrow = outrow = 0;
    while (outrow < cinfo->max_v_samp_factor) {
        /* Generate one output row with proper horizontal expansion */
        inptr = input_data[inrow];
        outptr = output_data[outrow];
        outend = outptr + cinfo->output_width;
        while (outptr < outend) {
            invalue = *inptr++; /* don't need GETJSAMPLE() here */
            for (h = h_expand; h > 0; h--) {
                *outptr++ = invalue;
            }
        }
        /* Generate any additional output rows by duplicating the first one */
        if (v_expand > 1) {
            jcopy_sample_rows(output_data, outrow, output_data, outrow+1,
                             v_expand-1, cinfo->output_width);
        }
        inrow++;
        outrow += v_expand;
    }
}

/*
 * Fast processing for the common case of 2:1 horizontal and 1:1 vertical.
 * It's still a box filter.
 */

METHODDEF(void)
h2v1_upsample (j_decompress_ptr cinfo, jpeg_component_info * comptr,
               JSAMPARRAY input_data, JSAMPARRAY * output_data_ptr)
{
    JSAMPARRAY output_data = *output_data_ptr;
    register JSAMPROW inptr, outptr;
    register JSAMPLE invalue;
    JSAMPROW outend;
    int inrow;

```



```

for (inrow = 0; inrow < cinfo->max_v_samp_factor; inrow++) {
    inptr = input_data[inrow];
    outptr = output_data[inrow];
    outend = outptr + cinfo->output_width;
    while (outptr < outend) {
        invalue = *inptr++; /* don't need GETJSAMPLE() here */
        *outptr++ = invalue;
        *outptr++ = invalue;
    }
}

/*
 * Fast processing for the common case of 2:1 horizontal and 2:1 vertical.
 * It's still a box filter.
 */

```

```

METHODDEF(void)
h2v2_upsample (j_decompress_ptr cinfo, jpeg_component_info * comptr,
               JSAMPARRAY input_data, JSAMPARRAY * output_data_ptr)
{
    JSAMPARRAY output_data = *output_data_ptr;
    register JSAMPROW inptr, outptr;
    register JSAMPLE invalue;
    JSAMPROW outend;
    int inrow, outrow;

    inrow = outrow = 0;
    while (outrow < cinfo->max_v_samp_factor) {
        inptr = input_data[inrow];
        outptr = output_data[outrow];
        outend = outptr + cinfo->output_width;
        while (outptr < outend) {
            invalue = *inptr++; /* don't need GETJSAMPLE() here */
            *outptr++ = invalue;
            *outptr++ = invalue;
        }
        jcopy_sample_rows(output_data, outrow, output_data, outrow+1,
                          1, cinfo->output_width);
        inrow++;
        outrow += 2;
    }
}

```

Fancy processing for the common case of 2:1 horizontal and 1:1 vertical.

The upsampling algorithm is linear interpolation between pixel centers, also known as a "triangle filter". This is a good compromise between speed and visual quality. The centers of the output pixels are 1/4 and 3/4 of the way between input pixel centers.

\*  
 \* A note about the "bias" calculations: when rounding fractional values to integer, we do not want to always round 0.5 up to the next integer.  
 \* If we did that, we'd introduce a noticeable bias towards larger values.  
 \* Instead, this code is arranged so that 0.5 will be rounded up or down at alternate pixel locations (a simple ordered dither pattern).  
 \*/

```

METHODDEF(void)
h2v1_fancy_upsample (j_decompress_ptr cinfo, jpeg_component_info * comptr,
                     JSAMPARRAY input_data, JSAMPARRAY * output_data_ptr)
{
    JSAMPARRAY output_data = *output_data_ptr;
    register JSAMPROW inptr, outptr;
    register int invalue;
    register JDIMENSION colctr;
    int inrow;

    for (inrow = 0; inrow < cinfo->max_v_samp_factor; inrow++) {
        inptr = input_data[inrow];
        outptr = output_data[inrow];
        /* Special case for first column */
        invalue = GETJSAMPLE(*inptr++);
        *outptr++ = (JSAMPLE) invalue;
        *outptr++ = (JSAMPLE) ((invalue * 3 + GETJSAMPLE(*inptr) + 2) >> 2);
    }
}

```

```

    for (colctr = compptr->downsampled_width - 2; colctr > 0; colctr--) {
        /* General case: 3/4 * nearer pixel + 1/4 * further pixel */
        invalue = GETJSAMPLE(*inptr++) * 3;
        *outptr++ = (JSAMPLE) ((invalue + GETJSAMPLE(inptr[-2]) + 1) >> 2);
        *outptr++ = (JSAMPLE) ((invalue + GETJSAMPLE(*inptr) + 2) >> 2);
    }

    /* Special case for last column */
    invalue = GETJSAMPLE(*inptr);
    *outptr++ = (JSAMPLE) ((invalue * 3 + GETJSAMPLE(inptr[-1]) + 1) >> 2);
    *outptr++ = (JSAMPLE) invalue;
}

/*
 * Fancy processing for the common case of 2:1 horizontal and 2:1 vertical.
 * Again a triangle filter; see comments for h2v1 case, above.
 *
 * It is OK for us to reference the adjacent input rows because we demanded
 * context from the main buffer controller (see initialization code).
 */

METHODDEF(void)
h2v2_fancy_upsample (j_decompress_ptr cinfo, jpeg_component_info * compptr,
                    JSAMPARRAY input_data, JSAMPARRAY * output_data_ptr)
{
    JSAMPARRAY output_data = *output_data_ptr;
    register JSAMPROW inptr0, inptr1, outptr;
    #if BITS_IN_JSAMPLE == 8
        register int thiscolsum, lastcolsum, nextcolsum;
    #else
        register INT32 thiscolsum, lastcolsum, nextcolsum;
    #endif
    register JDIMENSION colctr;
    int inrow, outrow, v;

    inrow = outrow = 0;
    while (outrow < cinfo->max_v_samp_factor) {
        for (v = 0; v < 2; v++) {
            /* inptr0 points to nearest input row, inptr1 points to next nearest */
            inptr0 = input_data[inrow];
            if (v == 0) /* next nearest is row above */
                inptr1 = input_data[inrow-1];
            else /* next nearest is row below */
                inptr1 = input_data[inrow+1];
            outptr = output_data[outrow++];

            /* Special case for first column */
            thiscolsum = GETJSAMPLE(*inptr0++) * 3 + GETJSAMPLE(*inptr1++);
            nextcolsum = GETJSAMPLE(*inptr0++) * 3 + GETJSAMPLE(*inptr1++);
            *outptr++ = (JSAMPLE) ((thiscolsum * 4 + 8) >> 4);
            *outptr++ = (JSAMPLE) ((thiscolsum * 3 + nextcolsum + 7) >> 4);
            lastcolsum = thiscolsum; thiscolsum = nextcolsum;

            for (colctr = compptr->downsampled_width - 2; colctr > 0; colctr--) {
                /* General case: 3/4 * nearer pixel + 1/4 * further pixel in each */
                /* dimension, thus 9/16, 3/16, 3/16, 1/16 overall */
                nextcolsum = GETJSAMPLE(*inptr0++) * 3 + GETJSAMPLE(*inptr1++);
                *outptr++ = (JSAMPLE) ((thiscolsum * 3 + lastcolsum + 8) >> 4);
                *outptr++ = (JSAMPLE) ((thiscolsum * 3 + nextcolsum + 7) >> 4);
                lastcolsum = thiscolsum; thiscolsum = nextcolsum;
            }

            /* Special case for last column */
            *outptr++ = (JSAMPLE) ((thiscolsum * 3 + lastcolsum + 8) >> 4);
            *outptr++ = (JSAMPLE) ((thiscolsum * 4 + 7) >> 4);
        }
        inrow++;
    }
}

/*
 * Module initialization routine for upsampling.
 */

GLOBAL(void)
jinit_upsampler (j_decompress_ptr cinfo)
{

```

```

my_upsample_ptr upsample;
int ci;
jpeg_component_info * compptr;
boolean need_buffer, do_fancy;
int h_in_group, v_in_group, h_out_group, v_out_group;

upsample = (my_upsample_ptr)
    (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
        SIZEOF(my_upsampler));
cinfo->upsample = (struct jpeg_upsampler *) upsample;
upsample->pub.start_pass = start_pass_upsample;
upsample->pub.upsample = sep_upsample;
upsample->pub.need_context_rows = FALSE; /* until we find out differently */

if (cinfo->CCIR601_sampling) /* this isn't supported */
    ERREXIT(cinfo, JERR_CCIR601_NOTIMPL);

/* jdmainct.c doesn't support context rows when min_DCT_scaled_size = 1,
 * so don't ask for it.
 */
do_fancy = cinfo->do_fancy_upsampling && cinfo->min_DCT_scaled_size > 1;

/* Verify we can handle the sampling factors, select per-component methods,
 * and create storage as needed.
 */
for (ci = 0, compptr = cinfo->comp_info; ci < cinfo->num_components;
    ci++, compptr++) {
    /* Compute size of an "input group" after IDCT scaling. This many samples
     * are to be converted to max_h_samp_factor * max_v_samp_factor pixels.
     */
    h_in_group = (compptr->h_samp_factor * compptr->DCT_scaled_size) /
        cinfo->min_DCT_scaled_size;
    v_in_group = (compptr->v_samp_factor * compptr->DCT_scaled_size) /
        cinfo->min_DCT_scaled_size;
    h_out_group = cinfo->max_h_samp_factor;
    v_out_group = cinfo->max_v_samp_factor;
    upsample->rowgroup_height[ci] = v_in_group; /* save for use later */
    need_buffer = TRUE;
    if (! compptr->component_needed) {
        /* Don't bother to upsample an uninteresting component. */
        upsample->methods[ci] = noop_upsample;
        need_buffer = FALSE;
    } else if (h_in_group == h_out_group && v_in_group == v_out_group) {
        /* Fullsize components can be processed without any work. */
        upsample->methods[ci] = fullsize_upsample;
        need_buffer = FALSE;
    } else if (h_in_group * 2 == h_out_group &&
        v_in_group == v_out_group) {
        /* Special cases for 2h1v upsampling */
        if (do_fancy && compptr->downsampled_width > 2)
            upsample->methods[ci] = h2v1_fancy_upsample;
        else
            upsample->methods[ci] = h2v1_upsample;
    } else if (h_in_group * 2 == h_out_group &&
        v_in_group * 2 == v_out_group) {
        /* Special cases for 2h2v upsampling */
        if (do_fancy && compptr->downsampled_width > 2) {
            upsample->methods[ci] = h2v2_fancy_upsample;
            upsample->pub.need_context_rows = TRUE;
        } else
            upsample->methods[ci] = h2v2_upsample;
    } else if ((h_out_group % h_in_group) == 0 &&
        (v_out_group % v_in_group) == 0) {
        /* Generic integral-factors upsampling method */
        upsample->methods[ci] = int_upsample;
        upsample->h_expand[ci] = (UINT8) (h_out_group / h_in_group);
        upsample->v_expand[ci] = (UINT8) (v_out_group / v_in_group);
    } else
        ERREXIT(cinfo, JERR_FRACT_SAMPLE_NOTIMPL);
    if (need_buffer) {
        upsample->color_buf[ci] = (*cinfo->mem->alloc_sarray)
            ((j_common_ptr) cinfo, JPOOL_IMAGE,
                (JDIMENSION) jround_up((long) cinfo->output_width,
                    (long) cinfo->max_h_samp_factor),
                (JDIMENSION) cinfo->max_v_samp_factor);
    }
}
}

```

```

/*
 * jdtrans.c
 *
 * Copyright (C) 1995-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains library routines for transcoding decompression,
 * that is, reading raw DCT coefficient arrays from an input JPEG file.
 * The routines in jdapimin.c will also be needed by a transcoder.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

/* Forward declarations */
LOCAL(void) transdecode_master_selection JPP((j_decompress_ptr cinfo));

/*
 * Read the coefficient arrays from a JPEG file.
 * jpeg_read_header must be completed before calling this.
 *
 * The entire image is read into a set of virtual coefficient-block arrays,
 * one per component. The return value is a pointer to the array of
 * virtual-array descriptors. These can be manipulated directly via the
 * JPEG memory manager, or handed off to jpeg_write_coefficients().
 * To release the memory occupied by the virtual arrays, call
 * jpeg_finish_decompress() when done with the data.
 *
 * An alternative usage is to simply obtain access to the coefficient arrays
 * during a buffered-image-mode decompression operation. This is allowed
 * after any jpeg_finish_output() call. The arrays can be accessed until
 * jpeg_finish_decompress() is called. (Note that any call to the library
 * may reposition the arrays, so don't rely on access_virt_barray() results
 * to stay valid across library calls.)
 *
 * Returns NULL if suspended. This case need be checked only if
 * a suspending data source is used.
 */

GLOBAL(jvirt_barray_ptr *)
jpeg_read_coefficients (j_decompress_ptr cinfo)
{
  if (cinfo->global_state == DSTATE_READY) {
    /* First call: initialize active modules */
    transdecode_master_selection(cinfo);
    cinfo->global_state = DSTATE_RDCOEFS;
  }
  if (cinfo->global_state == DSTATE_RDCOEFS) {
    /* Absorb whole file into the coef buffer */
    for (;;) {
      int retcode;
      /* Call progress monitor hook if present */
      if (cinfo->progress != NULL)
        (*cinfo->progress->progress_monitor) ((j_common_ptr) cinfo);
      /* Absorb some more input */
      retcode = (*cinfo->inputctl->consume_input) (cinfo);
      if (retcode == JPEG_SUSPENDED)
        return NULL;
      if (retcode == JPEG_REACHED_EOI)
        break;
      /* Advance progress counter if appropriate */
      if (cinfo->progress != NULL &&
          (retcode == JPEG_ROW_COMPLETED || retcode == JPEG_REACHED_SOS)) {
        if (++cinfo->progress->pass_counter >= cinfo->progress->pass_limit) {
          /* startup underestimated number of scans; ratchet up one scan */
          cinfo->progress->pass_limit += (long) cinfo->total_iMCU_rows;
        }
      }
    }
    /* Set state so that jpeg_finish_decompress does the right thing */
    cinfo->global_state = DSTATE_STOPPING;
  }
  /* At this point we should be in state DSTATE_STOPPING if being used
   * standalone, or in state DSTATE_BUFIMAGE if being invoked to get access
   * to the coefficients during a full buffered-image-mode decompression.
   */
}

```

```

    if ((cinfo->global_state == DSTATE_STOPPING ||
        cinfo->global_state == DSTATE_BUFIMAGE) && cinfo->buffered_image) {
        return cinfo->coef->coef_arrays;
    }
    /* Oops, improper usage */
    ERREXIT1(cinfo, JERR_BAD_STATE, cinfo->global_state);
    return NULL;          /* keep compiler happy */
}

/*
 * Master selection of decompression modules for transcoding.
 * This substitutes for jdmaster.c's initialization of the full decompressor.
 */

LOCAL(void)
transdecode_master_selection (j_decompress_ptr cinfo)
{
    /* This is effectively a buffered-image operation. */
    cinfo->buffered_image = TRUE;

    /* Entropy decoding: either Huffman or arithmetic coding. */
    if (cinfo->arith_code) {
        ERREXIT(cinfo, JERR_ARITH_NOTIMPL);
    } else {
        if (cinfo->progressive_mode) {
#ifdef D_PROGRESSIVE_SUPPORTED
            jinit_phuff_decoder(cinfo);
#else
            ERREXIT(cinfo, JERR_NOT_COMPILED);
#endif
        } else
            jinit_huff_decoder(cinfo);

        /* Always get a full-image coefficient buffer. */
        jinit_d_coef_controller(cinfo, TRUE);

        /* We can now tell the memory manager to allocate virtual arrays. */
        (*cinfo->mem->realize_virt_arrays) ((j_common_ptr) cinfo);

        /* Initialize input side of decompressor to consume first scan. */
        (*cinfo->inputctl->start_input_pass) (cinfo);

        /* Initialize progress monitoring. */
        if (cinfo->progress != NULL) {
            int nscans;
            /* Estimate number of scans to set pass_limit. */
            if (cinfo->progressive_mode) {
                /* Arbitrarily estimate 2 interleaved DC scans + 3 AC scans/component. */
                nscans = 2 + 3 * cinfo->num_components;
            } else if (cinfo->inputctl->has_multiple_scans) {
                /* For a nonprogressive multiscan file, estimate 1 scan per component. */
                nscans = cinfo->num_components;
            } else {
                nscans = 1;
            }
            cinfo->progress->pass_counter = 0L;
            cinfo->progress->pass_limit = (long) cinfo->total_iMCU_rows * nscans;
            cinfo->progress->completed_passes = 0;
            cinfo->progress->total_passes = 1;
        }
    }
}

```

```

/*
 * jerror.c
 *
 * Copyright (C) 1991-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains simple error-reporting and trace-message routines.
 * These are suitable for Unix-like systems and others where writing to
 * stderr is the right thing to do. Many applications will want to replace
 * some or all of these routines.
 *
 * If you define USE_WINDOWS_MESSAGEBOX in jconfig.h or in the makefile,
 * you get a Windows-specific hack to display error messages in a dialog box.
 * It ain't much, but it beats dropping error messages into the bit bucket,
 * which is what happens to output to stderr under most Windows C compilers.
 *
 * These routines are used by both the compression and decompression code.
 */

/* this is not a core library module, so it doesn't define JPEG_INTERNALS */
#include "jinclude.h"
#include "jpeglib.h"
#include "jversion.h"
#include "jerror.h"

#ifdef USE_WINDOWS_MESSAGEBOX
#include <windows.h>
#endif

#ifdef EXIT_FAILURE
/* define exit() codes if not provided */
#define EXIT_FAILURE 1
#endif

/*
 * Create the message string table.
 * We do this from the master message list in jerror.h by re-reading
 * jerror.h with a suitable definition for macro JMESSAGE.
 * The message table is made an external symbol just in case any applications
 * want to refer to it directly.
 */

#ifdef NEED_SHORT_EXTERNAL_NAMES
#define jpeg_std_message_table jMsgTable
#endif

#define JMESSAGE(code,string) string ,

const char * const jpeg_std_message_table[] = {
#include "jerror.h"
  NULL
};

/*
 * Error exit handler: must not return to caller.
 *
 * Applications may override this if they want to get control back after
 * an error. Typically one would longjmp somewhere instead of exiting.
 * The setjmp buffer can be made a private field within an expanded error
 * handler object. Note that the info needed to generate an error message
 * is stored in the error object, so you can generate the message now or
 * later, at your convenience.
 * You should make sure that the JPEG object is cleaned up (with jpeg_abort
 * or jpeg_destroy) at some point.
 */

METHODDEF(void)
error_exit (j_common_ptr cinfo)
{
  /* Always display the message */
  (*cinfo->err->output_message) (cinfo);

  /* Let the memory manager delete any temp files before we die */
  jpeg_destroy(cinfo);

  exit(EXIT_FAILURE);
}

```

```

/*
 * Actual output of an error or trace message.
 * Applications may override this method to send JPEG messages somewhere
 * other than stderr.
 *
 * On Windows, printing to stderr is generally completely useless,
 * so we provide optional code to produce an error-dialog popup.
 * Most Windows applications will still prefer to override this routine,
 * but if they don't, it'll do something at least marginally useful.
 *
 * NOTE: to use the library in an environment that doesn't support the
 * C stdio library, you may have to delete the call to fprintf() entirely,
 * not just not use this routine.
 */

METHODDEF(void)
output_message (j_common_ptr cinfo)
{
    char buffer[JMSG_LENGTH_MAX];

    /* Create the message */
    (*cinfo->err->format_message) (cinfo, buffer);

#ifdef USE_WINDOWS_MESSAGEBOX
    /* Display it in a message dialog box */
    MessageBox(GetActiveWindow(), buffer, "JPEG Library Error",
        MB_OK | MB_ICONERROR);
#else
    /* Send it to stderr, adding a newline */
    fprintf(stderr, "%s\n", buffer);
#endif
}

/*
 * Decide whether to emit a trace or warning message.
 * msg_level is one of:
 *   -1: recoverable corrupt-data warning, may want to abort.
 *   0: important advisory messages (always display to user).
 *   1: first level of tracing detail.
 *   2,3,...: successively more detailed tracing messages.
 * An application might override this method if it wanted to abort on warnings
 * or change the policy about which messages to display.
 */

METHODDEF(void)
emit_message (j_common_ptr cinfo, int msg_level)
{
    struct jpeg_error_mgr * err = cinfo->err;

    if (msg_level < 0) {
        /* It's a warning message. Since corrupt files may generate many warnings,
         * the policy implemented here is to show only the first warning,
         * unless trace_level >= 3.
         */
        if (err->num_warnings == 0 || err->trace_level >= 3)
            (*err->output_message) (cinfo);
        /* Always count warnings in num_warnings. */
        err->num_warnings++;
    } else {
        /* It's a trace message. Show it if trace_level >= msg_level. */
        if (err->trace_level >= msg_level)
            (*err->output_message) (cinfo);
    }
}

/*
 * Format a message string for the most recent JPEG error or message.
 * The message is stored into buffer, which should be at least JMSG_LENGTH_MAX
 * characters. Note that no '\n' character is added to the string.
 * Few applications should need to override this method.
 */

METHODDEF(void)
format_message (j_common_ptr cinfo, char * buffer)
{
    struct jpeg_error_mgr * err = cinfo->err;
    int msg_code = err->msg_code;

```

```

const char * msgtext = NULL;
const char * msgptr;
char ch;
boolean isstring;

/* Look up message string in proper table */
if (msg_code > 0 && msg_code <= err->last_jpeg_message) {
    msgtext = err->jpeg_message_table[msg_code];
} else if (err->addon_message_table != NULL &&
    msg_code >= err->first_addon_message &&
    msg_code <= err->last_addon_message) {
    msgtext = err->addon_message_table[msg_code - err->first_addon_message];
}

/* Defend against bogus message number */
if (msgtext == NULL) {
    err->msg_parm.i[0] = msg_code;
    msgtext = err->jpeg_message_table[0];
}

/* Check for string parameter, as indicated by %s in the message text */
isstring = FALSE;
msgptr = msgtext;
while ((ch = *msgptr++) != '\0') {
    if (ch == '%') {
        if (*msgptr == 's') isstring = TRUE;
        break;
    }
}

/* Format the message into the passed buffer */
if (isstring)
    sprintf(buffer, msgtext, err->msg_parm.s);
else
    sprintf(buffer, msgtext,
        err->msg_parm.i[0], err->msg_parm.i[1],
        err->msg_parm.i[2], err->msg_parm.i[3],
        err->msg_parm.i[4], err->msg_parm.i[5],
        err->msg_parm.i[6], err->msg_parm.i[7]);

/*
 * Reset error state variables at start of a new image.
 * This is called during compression startup to reset trace/error
 * processing to default state, without losing any application-specific
 * method pointers. An application might possibly want to override
 * this method if it has additional error processing state.
 */
METHODDEF(void)
reset_error_mgr (j_common_ptr cinfo)
{
    cinfo->err->num_warnings = 0;
    /* trace_level is not reset since it is an application-supplied parameter */
    cinfo->err->msg_code = 0; /* may be useful as a flag for "no error" */
}

/*
 * Fill in the standard error-handling methods in a jpeg_error_mgr object.
 * Typical call is:
 *   struct jpeg_compress_struct cinfo;
 *   struct jpeg_error_mgr err;
 *   cinfo.err = jpeg_std_error(&err);
 * after which the application may override some of the methods.
 */

GLOBAL(struct jpeg_error_mgr *)
jpeg_std_error (struct jpeg_error_mgr * err)
{
    err->error_exit = error_exit;
    err->emit_message = emit_message;
    err->output_message = output_message;
    err->format_message = format_message;
    err->reset_error_mgr = reset_error_mgr;

    err->trace_level = 0; /* default = no tracing */
    err->num_warnings = 0; /* no warnings emitted yet */
}

```



```

err->msg_code = 0;          /* may be useful as a flag for "no error" */

/* Initialize message table pointers */
err->jpeg_message_table = jpeg_std_message_table;
err->last_jpeg_message = (int) JMSG_LASTMSGCODE - 1;

err->addon_message_table = NULL;
err->first_addon_message = 0; /* for safety */
err->last_addon_message = 0;

return err;
}

```

```

/*
 * jfdctflt.c
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains a floating-point implementation of the
 * forward DCT (Discrete Cosine Transform).
 *
 * This implementation should be more accurate than either of the integer
 * DCT implementations. However, it may not give the same results on all
 * machines because of differences in roundoff behavior. Speed will depend
 * on the hardware's floating point capacity.
 *
 * A 2-D DCT can be done by 1-D DCT on each row followed by 1-D DCT
 * on each column. Direct algorithms are also available, but they are
 * much more complex and seem not to be any faster when reduced to code.
 *
 * This implementation is based on Arai, Agui, and Nakajima's algorithm for
 * scaled DCT. Their original paper (Trans. IEICE E-71(11):1095) is in
 * Japanese, but the algorithm is described in the Pennebaker & Mitchell
 * JPEG textbook (see REFERENCES section in file README). The following code
 * is based directly on figure 4-8 in P&M.
 * While an 8-point DCT cannot be done in less than 11 multiplies, it is
 * possible to arrange the computation so that many of the multiplies are
 * simple scalings of the final outputs. These multiplies can then be
 * folded into the multiplications or divisions by the JPEG quantization
 * table entries. The AA&N method leaves only 5 multiplies and 29 adds
 * to be done in the DCT itself.
 * The primary disadvantage of this method is that with a fixed-point
 * implementation, accuracy is lost due to imprecise representation of the
 * scaled quantization values. However, that problem does not arise if
 * we use floating point arithmetic.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jdct.h" /* Private declarations for DCT subsystem */

#ifdef DCT_FLOAT_SUPPORTED

/*
 * This module is specialized to the case DCTSIZE = 8.
 */

#if DCTSIZE != 8
  Sorry, this code only copes with 8x8 DCTs. /* deliberate syntax err */
#endif

/*
 * Perform the forward DCT on one block of samples.
 */

GLOBAL(void)
jpeg_fdct_float (FAST_FLOAT * data)
{
  FAST_FLOAT tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;
  FAST_FLOAT tmp10, tmp11, tmp12, tmp13;
  FAST_FLOAT z1, z2, z3, z4, z5, z11, z13;
  FAST_FLOAT *dataptr;
  int ctr;

  /* Pass 1: process rows. */

  dataptr = data;
  for (ctr = DCTSIZE-1; ctr >= 0; ctr--) {
    tmp0 = dataptr[0] + dataptr[7];
    tmp7 = dataptr[0] - dataptr[7];
    tmp1 = dataptr[1] + dataptr[6];
    tmp6 = dataptr[1] - dataptr[6];
    tmp2 = dataptr[2] + dataptr[5];
    tmp5 = dataptr[2] - dataptr[5];
    tmp3 = dataptr[3] + dataptr[4];
    tmp4 = dataptr[3] - dataptr[4];

    /* Even part */

```

```

tmp10 = tmp0 + tmp3;    /* phase 2 */
tmp13 = tmp0 - tmp3;
tmp11 = tmp1 + tmp2;
tmp12 = tmp1 - tmp2;

dataptr[0] = tmp10 + tmp11; /* phase 3 */
dataptr[4] = tmp10 - tmp11;

z1 = (tmp12 + tmp13) * ((FAST_FLOAT) 0.707106781); /* c4 */
dataptr[2] = tmp13 + z1;    /* phase 5 */
dataptr[6] = tmp13 - z1;

/* Odd part */

tmp10 = tmp4 + tmp5;    /* phase 2 */
tmp11 = tmp5 + tmp6;
tmp12 = tmp6 + tmp7;

/* The rotator is modified from fig 4-8 to avoid extra negations. */
z5 = (tmp10 - tmp12) * ((FAST_FLOAT) 0.382683433); /* c6 */
z2 = ((FAST_FLOAT) 0.541196100) * tmp10 + z5; /* c2-c6 */
z4 = ((FAST_FLOAT) 1.306562965) * tmp12 + z5; /* c2+c6 */
z3 = tmp11 * ((FAST_FLOAT) 0.707106781); /* c4 */

z11 = tmp7 + z3;        /* phase 5 */
z13 = tmp7 - z3;

dataptr[5] = z13 + z2;  /* phase 6 */
dataptr[3] = z13 - z2;
dataptr[1] = z11 + z4;
dataptr[7] = z11 - z4;

dataptr += DCTSIZE;    /* advance pointer to next row */

* Pass 2: process columns. */
dataptr = data;
for (ctr = DCTSIZE-1; ctr >= 0; ctr--) {
    tmp0 = dataptr[DCTSIZE*0] + dataptr[DCTSIZE*7];
    tmp7 = dataptr[DCTSIZE*0] - dataptr[DCTSIZE*7];
    tmp1 = dataptr[DCTSIZE*1] + dataptr[DCTSIZE*6];
    tmp6 = dataptr[DCTSIZE*1] - dataptr[DCTSIZE*6];
    tmp2 = dataptr[DCTSIZE*2] + dataptr[DCTSIZE*5];
    tmp5 = dataptr[DCTSIZE*2] - dataptr[DCTSIZE*5];
    tmp3 = dataptr[DCTSIZE*3] + dataptr[DCTSIZE*4];
    tmp4 = dataptr[DCTSIZE*3] - dataptr[DCTSIZE*4];

    /* Even part */

    tmp10 = tmp0 + tmp3;    /* phase 2 */
    tmp13 = tmp0 - tmp3;
    tmp11 = tmp1 + tmp2;
    tmp12 = tmp1 - tmp2;

    dataptr[DCTSIZE*0] = tmp10 + tmp11; /* phase 3 */
    dataptr[DCTSIZE*4] = tmp10 - tmp11;

    z1 = (tmp12 + tmp13) * ((FAST_FLOAT) 0.707106781); /* c4 */
    dataptr[DCTSIZE*2] = tmp13 + z1; /* phase 5 */
    dataptr[DCTSIZE*6] = tmp13 - z1;

    /* Odd part */

    tmp10 = tmp4 + tmp5;    /* phase 2 */
    tmp11 = tmp5 + tmp6;
    tmp12 = tmp6 + tmp7;

    /* The rotator is modified from fig 4-8 to avoid extra negations. */
    z5 = (tmp10 - tmp12) * ((FAST_FLOAT) 0.382683433); /* c6 */
    z2 = ((FAST_FLOAT) 0.541196100) * tmp10 + z5; /* c2-c6 */
    z4 = ((FAST_FLOAT) 1.306562965) * tmp12 + z5; /* c2+c6 */
    z3 = tmp11 * ((FAST_FLOAT) 0.707106781); /* c4 */

    z11 = tmp7 + z3;        /* phase 5 */
    z13 = tmp7 - z3;

    dataptr[DCTSIZE*5] = z13 + z2; /* phase 6 */
    dataptr[DCTSIZE*3] = z13 - z2;

```

```

    dataptr[DCTSIZE*1] = z11 + z4;
    dataptr[DCTSIZE*7] = z11 - z4;

    dataptr++;          /* advance pointer to next column */
}
#endif /* DCT_FLOAT_SUPPORTED */

```

```

/*
 * jfdctfst.c
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains a fast, not so accurate integer implementation of the
 * forward DCT (Discrete Cosine Transform).
 *
 * A 2-D DCT can be done by 1-D DCT on each row followed by 1-D DCT
 * on each column. Direct algorithms are also available, but they are
 * much more complex and seem not to be any faster when reduced to code.
 *
 * This implementation is based on Arai, Agui, and Nakajima's algorithm for
 * scaled DCT. Their original paper (Trans. IEICE E-71(11):1095) is in
 * Japanese, but the algorithm is described in the Pennebaker & Mitchell
 * JPEG textbook (see REFERENCES section in file README). The following code
 * is based directly on figure 4-8 in P&M.
 * While an 8-point DCT cannot be done in less than 11 multiplies, it is
 * possible to arrange the computation so that many of the multiplies are
 * simple scalings of the final outputs. These multiplies can then be
 * folded into the multiplications or divisions by the JPEG quantization
 * table entries. The AA&N method leaves only 5 multiplies and 29 adds
 * to be done in the DCT itself.
 * The primary disadvantage of this method is that with fixed-point math,
 * accuracy is lost due to imprecise representation of the scaled
 * quantization values. The smaller the quantization table entry, the less
 * precise the scaled value, so this implementation does worse with high-
 * quality-setting files than with low-quality ones.
 */
#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jdct.h" /* Private declarations for DCT subsystem */

#ifdef DCT_IFAST_SUPPORTED

/*
 * This module is specialized to the case DCTSIZE = 8.
 */

#if DCTSIZE != 8
  Sorry, this code only copes with 8x8 DCTs. /* deliberate syntax err */
#endif

/*
 * Scaling decisions are generally the same as in the LL&M algorithm;
 * see jfdctint.c for more details. However, we choose to descale
 * (right shift) multiplication products as soon as they are formed,
 * rather than carrying additional fractional bits into subsequent additions.
 * This compromises accuracy slightly, but it lets us save a few shifts.
 * More importantly, 16-bit arithmetic is then adequate (for 8-bit samples)
 * everywhere except in the multiplications proper; this saves a good deal
 * of work on 16-bit-int machines.
 *
 * Again to save a few shifts, the intermediate results between pass 1 and
 * pass 2 are not upscaled, but are represented only to integral precision.
 *
 * A final compromise is to represent the multiplicative constants to only
 * 8 fractional bits, rather than 13. This saves some shifting work on some
 * machines, and may also reduce the cost of multiplication (since there
 * are fewer one-bits in the constants).
 */

#define CONST_BITS 8

/*
 * Some C compilers fail to reduce "FIX(constant)" at compile time, thus
 * causing a lot of useless floating-point operations at run time.
 * To get around this we use the following pre-calculated constants.
 * If you change CONST_BITS you may want to add appropriate values.
 * (With a reasonable C compiler, you can just rely on the FIX() macro...)
 */

#if CONST_BITS == 8
#define FIX_0_382683433 ((INT32) 98) /* FIX(0.382683433) */
#define FIX_0_541196100 ((INT32) 139) /* FIX(0.541196100) */

```

```

#define FIX_0_707106781 ((INT32) 181) /* FIX(0.707106781) */
#define FIX_1_306562965 ((INT32) 334) /* FIX(1.306562965) */
#else
#define FIX_0_382683433 FIX(0.382683433)
#define FIX_0_541196100 FIX(0.541196100)
#define FIX_0_707106781 FIX(0.707106781)
#define FIX_1_306562965 FIX(1.306562965)
#endif

/* We can gain a little more speed, with a further compromise in accuracy,
 * by omitting the addition in a descaling shift. This yields an incorrectly
 * rounded result half the time...
 */

#ifndef USE_ACCURATE_ROUNDING
#undef DESCALE
#define DESCALE(x,n) RIGHT_SHIFT(x, n)
#endif

/* Multiply a DCTELEM variable by an INT32 constant, and immediately
 * descale to yield a DCTELEM result.
 */

#define MULTIPLY(var,const) ((DCTELEM) DESCALE((var) * (const), CONST_BITS))

/*
 * Perform the forward DCT on one block of samples.
 */
GLOBAL(void)
jpeg_fdct_ifast (DCTELEM * data)
{
    DCTELEM tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;
    DCTELEM tmp10, tmp11, tmp12, tmp13;
    DCTELEM z1, z2, z3, z4, z5, z11, z13;
    DCTELEM *dataptr;
    int ctr;
    SHIFT_TEMPS

    /* Pass 1: process rows. */

    dataptr = data;
    for (ctr = DCTSIZE-1; ctr >= 0; ctr--) {
        tmp0 = dataptr[0] + dataptr[7];
        tmp7 = dataptr[0] - dataptr[7];
        tmp1 = dataptr[1] + dataptr[6];
        tmp6 = dataptr[1] - dataptr[6];
        tmp2 = dataptr[2] + dataptr[5];
        tmp5 = dataptr[2] - dataptr[5];
        tmp3 = dataptr[3] + dataptr[4];
        tmp4 = dataptr[3] - dataptr[4];

        /* Even part */

        tmp10 = tmp0 + tmp3; /* phase 2 */
        tmp13 = tmp0 - tmp3;
        tmp11 = tmp1 + tmp2;
        tmp12 = tmp1 - tmp2;

        dataptr[0] = tmp10 + tmp11; /* phase 3 */
        dataptr[4] = tmp10 - tmp11;

        z1 = MULTIPLY(tmp12 + tmp13, FIX_0_707106781); /* c4 */
        dataptr[2] = tmp13 + z1; /* phase 5 */
        dataptr[6] = tmp13 - z1;

        /* Odd part */

        tmp10 = tmp4 + tmp5; /* phase 2 */
        tmp11 = tmp5 + tmp6;
        tmp12 = tmp6 + tmp7;

        /* The rotator is modified from fig 4-8 to avoid extra negations. */
        z5 = MULTIPLY(tmp10 - tmp12, FIX_0_382683433); /* c6 */
        z2 = MULTIPLY(tmp10, FIX_0_541196100) + z5; /* c2-c6 */
        z4 = MULTIPLY(tmp12, FIX_1_306562965) + z5; /* c2+c6 */
        z3 = MULTIPLY(tmp11, FIX_0_707106781); /* c4 */

```

```

z11 = tmp7 + z3;          /* phase 5 */
z13 = tmp7 - z3;

dataptr[5] = z13 + z2;    /* phase 6 */
dataptr[3] = z13 - z2;
dataptr[1] = z11 + z4;
dataptr[7] = z11 - z4;

dataptr += DCTSIZE;      /* advance pointer to next row */
}

/* Pass 2: process columns. */

dataptr = data;
for (ctr = DCTSIZE-1; ctr >= 0; ctr--) {
    tmp0 = dataptr[DCTSIZE*0] + dataptr[DCTSIZE*7];
    tmp7 = dataptr[DCTSIZE*0] - dataptr[DCTSIZE*7];
    tmp1 = dataptr[DCTSIZE*1] + dataptr[DCTSIZE*6];
    tmp6 = dataptr[DCTSIZE*1] - dataptr[DCTSIZE*6];
    tmp2 = dataptr[DCTSIZE*2] + dataptr[DCTSIZE*5];
    tmp5 = dataptr[DCTSIZE*2] - dataptr[DCTSIZE*5];
    tmp3 = dataptr[DCTSIZE*3] + dataptr[DCTSIZE*4];
    tmp4 = dataptr[DCTSIZE*3] - dataptr[DCTSIZE*4];

    /* Even part */

    tmp10 = tmp0 + tmp3;    /* phase 2 */
    tmp13 = tmp0 - tmp3;
    tmp11 = tmp1 + tmp2;
    tmp12 = tmp1 - tmp2;

    dataptr[DCTSIZE*0] = tmp10 + tmp11; /* phase 3 */
    dataptr[DCTSIZE*4] = tmp10 - tmp11;

    z1 = MULTIPLY(tmp12 + tmp13, FIX_0_707106781); /* c4 */
    dataptr[DCTSIZE*2] = tmp13 + z1; /* phase 5 */
    dataptr[DCTSIZE*6] = tmp13 - z1;

    /* Odd part */

    tmp10 = tmp4 + tmp5;    /* phase 2 */
    tmp11 = tmp5 + tmp6;
    tmp12 = tmp6 + tmp7;

    /* The rotator is modified from fig 4-8 to avoid extra negations. */
    z5 = MULTIPLY(tmp10 - tmp12, FIX_0_382683433); /* c6 */
    z2 = MULTIPLY(tmp10, FIX_0_541196100) + z5; /* c2-c6 */
    z4 = MULTIPLY(tmp12, FIX_1_306562965) + z5; /* c2+c6 */
    z3 = MULTIPLY(tmp11, FIX_0_707106781); /* c4 */

    z11 = tmp7 + z3;        /* phase 5 */
    z13 = tmp7 - z3;

    dataptr[DCTSIZE*5] = z13 + z2; /* phase 6 */
    dataptr[DCTSIZE*3] = z13 - z2;
    dataptr[DCTSIZE*1] = z11 + z4;
    dataptr[DCTSIZE*7] = z11 - z4;

    dataptr++;              /* advance pointer to next column */
}
}

#endif /* DCT_IFAST_SUPPORTED */

```

```

/*
 * jfdctint.c
 *
 * Copyright (C) 1991-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains a slow-but-accurate integer implementation of the
 * forward DCT (Discrete Cosine Transform).
 *
 * A 2-D DCT can be done by 1-D DCT on each row followed by 1-D DCT
 * on each column. Direct algorithms are also available, but they are
 * much more complex and seem not to be any faster when reduced to code.
 *
 * This implementation is based on an algorithm described in
 * C. Loeffler, A. Ligtenberg and G. Moschytz, "Practical Fast 1-D DCT
 * Algorithms with 11 Multiplications", Proc. Int'l. Conf. on Acoustics,
 * Speech, and Signal Processing 1989 (ICASSP '89), pp. 988-991.
 * The primary algorithm described there uses 11 multiplies and 29 adds.
 * We use their alternate method with 12 multiplies and 32 adds.
 * The advantage of this method is that no data path contains more than one
 * multiplication; this allows a very simple and accurate implementation in
 * scaled fixed-point arithmetic, with a minimal number of shifts.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jdst.h" /* Private declarations for DCT subsystem */

#ifdef DCT_ISLOW_SUPPORTED

/*
 * This module is specialized to the case DCTSIZE = 8.
 */

#if DCTSIZE != 8
  Sorry, this code only copes with 8x8 DCTs. /* deliberate syntax err */
#endif

/*
 * The poop on this scaling stuff is as follows:
 *
 * Each 1-D DCT step produces outputs which are a factor of sqrt(N)
 * larger than the true DCT outputs. The final outputs are therefore
 * a factor of N larger than desired; since N=8 this can be cured by
 * a simple right shift at the end of the algorithm. The advantage of
 * this arrangement is that we save two multiplications per 1-D DCT,
 * because the y0 and y4 outputs need not be divided by sqrt(N).
 * In the IJG code, this factor of 8 is removed by the quantization step
 * (in jcdctmgr.c), NOT in this module.
 *
 * We have to do addition and subtraction of the integer inputs, which
 * is no problem, and multiplication by fractional constants, which is
 * a problem to do in integer arithmetic. We multiply all the constants
 * by CONST_SCALE and convert them to integer constants (thus retaining
 * CONST_BITS bits of precision in the constants). After doing a
 * multiplication we have to divide the product by CONST_SCALE, with proper
 * rounding, to produce the correct output. This division can be done
 * cheaply as a right shift of CONST_BITS bits. We postpone shifting
 * as long as possible so that partial sums can be added together with
 * full fractional precision.
 *
 * The outputs of the first pass are scaled up by PASS1_BITS bits so that
 * they are represented to better-than-integral precision. These outputs
 * require BITS_IN_JSAMPLE + PASS1_BITS + 3 bits; this fits in a 16-bit word
 * with the recommended scaling. (For 12-bit sample data, the intermediate
 * array is INT32 anyway.)
 *
 * To avoid overflow of the 32-bit intermediate results in pass 2, we must
 * have BITS_IN_JSAMPLE + CONST_BITS + PASS1_BITS <= 26. Error analysis
 * shows that the values given below are the most effective.
 */

#if BITS_IN_JSAMPLE == 8
#define CONST_BITS 13
#define PASS1_BITS 2
#else

```



```

#define CONST_BITS 13
#define PASS1_BITS 1 /* lose a little precision to avoid overflow */
#endif

/* Some C compilers fail to reduce "FIX(constant)" at compile time, thus
 * causing a lot of useless floating-point operations at run time.
 * To get around this we use the following pre-calculated constants.
 * If you change CONST_BITS you may want to add appropriate values.
 * (With a reasonable C compiler, you can just rely on the FIX() macro...)
 */

#if CONST_BITS == 13
#define FIX_0_298631336 ((INT32) 2446) /* FIX(0.298631336) */
#define FIX_0_390180644 ((INT32) 3196) /* FIX(0.390180644) */
#define FIX_0_541196100 ((INT32) 4433) /* FIX(0.541196100) */
#define FIX_0_765366865 ((INT32) 6270) /* FIX(0.765366865) */
#define FIX_0_899976223 ((INT32) 7373) /* FIX(0.899976223) */
#define FIX_1_175875602 ((INT32) 9633) /* FIX(1.175875602) */
#define FIX_1_501321110 ((INT32) 12299) /* FIX(1.501321110) */
#define FIX_1_847759065 ((INT32) 15137) /* FIX(1.847759065) */
#define FIX_1_961570560 ((INT32) 16069) /* FIX(1.961570560) */
#define FIX_2_053119869 ((INT32) 16819) /* FIX(2.053119869) */
#define FIX_2_562915447 ((INT32) 20995) /* FIX(2.562915447) */
#define FIX_3_072711026 ((INT32) 25172) /* FIX(3.072711026) */
#else
#define FIX_0_298631336 FIX(0.298631336)
#define FIX_0_390180644 FIX(0.390180644)
#define FIX_0_541196100 FIX(0.541196100)
#define FIX_0_765366865 FIX(0.765366865)
#define FIX_0_899976223 FIX(0.899976223)
#define FIX_1_175875602 FIX(1.175875602)
#define FIX_1_501321110 FIX(1.501321110)
#define FIX_1_847759065 FIX(1.847759065)
#define FIX_1_961570560 FIX(1.961570560)
#define FIX_2_053119869 FIX(2.053119869)
#define FIX_2_562915447 FIX(2.562915447)
#define FIX_3_072711026 FIX(3.072711026)
#endif

/* Multiply an INT32 variable by an INT32 constant to yield an INT32 result.
 * For 8-bit samples with the recommended scaling, all the variable
 * and constant values involved are no more than 16 bits wide, so a
 * 16x16->32 bit multiply can be used instead of a full 32x32 multiply.
 * For 12-bit samples, a full 32-bit multiplication will be needed.
 */

#if BITS_IN_JSAMPLE == 8
#define MULTIPLY(var,const) MULTIPLY16C16(var,const)
#else
#define MULTIPLY(var,const) ((var) * (const))
#endif

/*
 * Perform the forward DCT on one block of samples.
 */

GLOBAL(void)
jpeg_fdct_islow (DCTELEM * data)
{
    INT32 tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;
    INT32 tmp10, tmp11, tmp12, tmp13;
    INT32 z1, z2, z3, z4, z5;
    DCTELEM *dataptr;
    int ctr;
    SHIFT_TEMPS

    /* Pass 1: process rows. */
    /* Note results are scaled up by sqrt(8) compared to a true DCT; */
    /* furthermore, we scale the results by 2**PASS1_BITS. */

    dataptr = data;
    for (ctr = DCTSIZE-1; ctr >= 0; ctr--) {
        tmp0 = dataptr[0] + dataptr[7];
        tmp7 = dataptr[0] - dataptr[7];
        tmp1 = dataptr[1] + dataptr[6];
        tmp6 = dataptr[1] - dataptr[6];
        tmp2 = dataptr[2] + dataptr[5];
        tmp5 = dataptr[2] - dataptr[5];

```

```

tmp3 = dataptr[3] + dataptr[4];
tmp4 = dataptr[3] - dataptr[4];

/* Even part per LL&M figure 1 --- note that published figure is faulty;
 * rotator "sqrt(2)*c1" should be "sqrt(2)*c6".
 */

tmp10 = tmp0 + tmp3;
tmp13 = tmp0 - tmp3;
tmp11 = tmp1 + tmp2;
tmp12 = tmp1 - tmp2;

dataptr[0] = (DCTELEM) ((tmp10 + tmp11) << PASS1_BITS);
dataptr[4] = (DCTELEM) ((tmp10 - tmp11) << PASS1_BITS);

z1 = MULTIPLY(tmp12 + tmp13, FIX_0_541196100);
dataptr[2] = (DCTELEM) DESCALE(z1 + MULTIPLY(tmp13, FIX_0_765366865),
CONST_BITS-PASS1_BITS);
dataptr[6] = (DCTELEM) DESCALE(z1 + MULTIPLY(tmp12, - FIX_1_847759065),
CONST_BITS-PASS1_BITS);

/* Odd part per figure 8 --- note paper omits factor of sqrt(2).
 * cK represents cos(K*pi/16).
 * i0..i3 in the paper are tmp4..tmp7 here.
 */

z1 = tmp4 + tmp7;
z2 = tmp5 + tmp6;
z3 = tmp4 + tmp6;
z4 = tmp5 + tmp7;
z5 = MULTIPLY(z3 + z4, FIX_1_175875602); /* sqrt(2) * c3 */

tmp4 = MULTIPLY(tmp4, FIX_0_298631336); /* sqrt(2) * (-c1+c3+c5-c7) */
tmp5 = MULTIPLY(tmp5, FIX_2_053119869); /* sqrt(2) * ( c1+c3-c5+c7) */
tmp6 = MULTIPLY(tmp6, FIX_3_072711026); /* sqrt(2) * ( c1+c3+c5-c7) */
tmp7 = MULTIPLY(tmp7, FIX_1_501321110); /* sqrt(2) * ( c1+c3-c5-c7) */
z1 = MULTIPLY(z1, - FIX_0_899976223); /* sqrt(2) * (c7-c3) */
z2 = MULTIPLY(z2, - FIX_2_562915447); /* sqrt(2) * (-c1-c3) */
z3 = MULTIPLY(z3, - FIX_1_961570560); /* sqrt(2) * (-c3-c5) */
z4 = MULTIPLY(z4, - FIX_0_390180644); /* sqrt(2) * (c5-c3) */

z3 += z5;
z4 += z5;

dataptr[7] = (DCTELEM) DESCALE(tmp4 + z1 + z3, CONST_BITS-PASS1_BITS);
dataptr[5] = (DCTELEM) DESCALE(tmp5 + z2 + z4, CONST_BITS-PASS1_BITS);
dataptr[3] = (DCTELEM) DESCALE(tmp6 + z2 + z3, CONST_BITS-PASS1_BITS);
dataptr[1] = (DCTELEM) DESCALE(tmp7 + z1 + z4, CONST_BITS-PASS1_BITS);

dataptr += DCTSIZE; /* advance pointer to next row */

/* Pass 2: process columns.
 * We remove the PASS1_BITS scaling, but leave the results scaled up
 * by an overall factor of 8.
 */

dataptr = data;
for (ctr = DCTSIZE-1; ctr >= 0; ctr--) {
    tmp0 = dataptr[DCTSIZE*0] + dataptr[DCTSIZE*7];
    tmp7 = dataptr[DCTSIZE*0] - dataptr[DCTSIZE*7];
    tmp1 = dataptr[DCTSIZE*1] + dataptr[DCTSIZE*6];
    tmp6 = dataptr[DCTSIZE*1] - dataptr[DCTSIZE*6];
    tmp2 = dataptr[DCTSIZE*2] + dataptr[DCTSIZE*5];
    tmp5 = dataptr[DCTSIZE*2] - dataptr[DCTSIZE*5];
    tmp3 = dataptr[DCTSIZE*3] + dataptr[DCTSIZE*4];
    tmp4 = dataptr[DCTSIZE*3] - dataptr[DCTSIZE*4];

    /* Even part per LL&M figure 1 --- note that published figure is faulty;
     * rotator "sqrt(2)*c1" should be "sqrt(2)*c6".
     */

    tmp10 = tmp0 + tmp3;
    tmp13 = tmp0 - tmp3;
    tmp11 = tmp1 + tmp2;
    tmp12 = tmp1 - tmp2;

    dataptr[DCTSIZE*0] = (DCTELEM) DESCALE(tmp10 + tmp11, PASS1_BITS);
    dataptr[DCTSIZE*4] = (DCTELEM) DESCALE(tmp10 - tmp11, PASS1_BITS);

```

```

z1 = MULTIPLY(tmp12 + tmp13, FIX_0_541196100);
dataptr[DCTSIZE*2] = (DCTELEM) DESCALE(z1 + MULTIPLY(tmp13, FIX_0_765366865),
CONST_BITS+PASS1_BITS);
dataptr[DCTSIZE*6] = (DCTELEM) DESCALE(z1 + MULTIPLY(tmp12, - FIX_1_847759065),
CONST_BITS+PASS1_BITS);

/* Odd part per figure 8 --- note paper omits factor of sqrt(2).
* cK represents cos(K*pi/16).
* i0..i3 in the paper are tmp4..tmp7 here.
*/

z1 = tmp4 + tmp7;
z2 = tmp5 + tmp6;
z3 = tmp4 + tmp6;
z4 = tmp5 + tmp7;
z5 = MULTIPLY(z3 + z4, FIX_1_175875602); /* sqrt(2) * c3 */

tmp4 = MULTIPLY(tmp4, FIX_0_298631336); /* sqrt(2) * (-c1+c3+c5-c7) */
tmp5 = MULTIPLY(tmp5, FIX_2_053119869); /* sqrt(2) * ( c1+c3-c5+c7) */
tmp6 = MULTIPLY(tmp6, FIX_3_072711026); /* sqrt(2) * ( c1+c3+c5-c7) */
tmp7 = MULTIPLY(tmp7, FIX_1_501321110); /* sqrt(2) * ( c1+c3-c5-c7) */
z1 = MULTIPLY(z1, - FIX_0_899976223); /* sqrt(2) * (c7-c3) */
z2 = MULTIPLY(z2, - FIX_2_562915447); /* sqrt(2) * (-c1-c3) */
z3 = MULTIPLY(z3, - FIX_1_961570560); /* sqrt(2) * (-c3-c5) */
z4 = MULTIPLY(z4, - FIX_0_390180644); /* sqrt(2) * (c5-c3) */

z3 += z5;
z4 += z5;

dataptr[DCTSIZE*7] = (DCTELEM) DESCALE(tmp4 + z1 + z3,
CONST_BITS+PASS1_BITS);
dataptr[DCTSIZE*5] = (DCTELEM) DESCALE(tmp5 + z2 + z4,
CONST_BITS+PASS1_BITS);
dataptr[DCTSIZE*3] = (DCTELEM) DESCALE(tmp6 + z2 + z3,
CONST_BITS+PASS1_BITS);
dataptr[DCTSIZE*1] = (DCTELEM) DESCALE(tmp7 + z1 + z4,
CONST_BITS+PASS1_BITS);

dataptr++; /* advance pointer to next column */
#endif /* DCT_ISLOW_SUPPORTED */

```

```

/*
 * jidctflt.c
 *
 * Copyright (C) 1994-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains a floating-point implementation of the
 * inverse DCT (Discrete Cosine Transform).  In the IJG code, this routine
 * must also perform dequantization of the input coefficients.
 *
 * This implementation should be more accurate than either of the integer
 * IDCT implementations.  However, it may not give the same results on all
 * machines because of differences in roundoff behavior.  Speed will depend
 * on the hardware's floating point capacity.
 *
 * A 2-D IDCT can be done by 1-D IDCT on each column followed by 1-D IDCT
 * on each row (or vice versa, but it's more convenient to emit a row at
 * a time).  Direct algorithms are also available, but they are much more
 * complex and seem not to be any faster when reduced to code.
 *
 * This implementation is based on Arai, Agui, and Nakajima's algorithm for
 * scaled DCT.  Their original paper (Trans. IEICE E-71(11):1095) is in
 * Japanese, but the algorithm is described in the Pennebaker & Mitchell
 * JPEG textbook (see REFERENCES section in file README).  The following code
 * is based directly on figure 4-8 in P&M.
 * While an 8-point DCT cannot be done in less than 11 multiplies, it is
 * possible to arrange the computation so that many of the multiplies are
 * simple scalings of the final outputs.  These multiplies can then be
 * folded into the multiplications or divisions by the JPEG quantization
 * table entries.  The AA&N method leaves only 5 multiplies and 29 adds
 * to be done in the DCT itself.
 * The primary disadvantage of this method is that with a fixed-point
 * implementation, accuracy is lost due to imprecise representation of the
 * scaled quantization values.  However, that problem does not arise if
 * we use floating point arithmetic.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jdct.h"          /* Private declarations for DCT subsystem */

#ifdef DCT_FLOAT_SUPPORTED

/* This module is specialized to the case DCTSIZE = 8. */

#if DCTSIZE != 8
  Sorry, this code only copes with 8x8 DCTs. /* deliberate syntax err */
#endif

/* Dequantize a coefficient by multiplying it by the multiplier-table
 * entry; produce a float result.
 */

#define DEQUANTIZE(coef,quantval)  (((FAST_FLOAT) (coef)) * (quantval))

/*
 * Perform dequantization and inverse DCT on one block of coefficients.
 */

GLOBAL(void)
jpeg_idct_float (j_decompress_ptr cinfo, jpeg_component_info * comp_ptr,
                 JCOEFPTR coef_block,
                 JSAMPARRAY output_buf, JDIMENSION output_col)
{
  FAST_FLOAT tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;
  FAST_FLOAT tmp10, tmp11, tmp12, tmp13;
  FAST_FLOAT z5, z10, z11, z12, z13;
  JCOEFPTR in_ptr;
  FLOAT_MULT_TYPE * quant_ptr;
  FAST_FLOAT * wsptr;
  JSAMPROW out_ptr;
  JSAMPLE *range_limit = IDCT_range_limit(cinfo);
  int ctr;


```

```

FAST_FLOAT workspace[DCTSIZE2]; /* buffers data between passes */
SHIFT_TEMPS

/* Pass 1: process columns from input, store into work array. */

inptr = coef_block;
quantptr = (FLOAT_MULT_TYPE *) compptr-> dct_table;
wsptr = workspace;
for (ctr = DCTSIZE; ctr > 0; ctr--) {
    /* Due to quantization, we will usually find that many of the input
     * coefficients are zero, especially the AC terms. We can exploit this
     * by short-circuiting the IDCT calculation for any column in which all
     * the AC terms are zero. In that case each output is equal to the
     * DC coefficient (with scale factor as needed).
     * With typical images and quantization tables, half or more of the
     * column DCT calculations can be simplified this way.
     */

    if (inptr[DCTSIZE*1] == 0 && inptr[DCTSIZE*2] == 0 &&
        inptr[DCTSIZE*3] == 0 && inptr[DCTSIZE*4] == 0 &&
        inptr[DCTSIZE*5] == 0 && inptr[DCTSIZE*6] == 0 &&
        inptr[DCTSIZE*7] == 0) {
        /* AC terms all zero */
        FAST_FLOAT dval = DEQUANTIZE(inptr[DCTSIZE*0], quantptr[DCTSIZE*0]);

        wsptr[DCTSIZE*0] = dval;
        wsptr[DCTSIZE*1] = dval;
        wsptr[DCTSIZE*2] = dval;
        wsptr[DCTSIZE*3] = dval;
        wsptr[DCTSIZE*4] = dval;
        wsptr[DCTSIZE*5] = dval;
        wsptr[DCTSIZE*6] = dval;
        wsptr[DCTSIZE*7] = dval;

        inptr++;
        quantptr++;
        wsptr++;
        continue;
    }

    /* Even part */

    tmp0 = DEQUANTIZE(inptr[DCTSIZE*0], quantptr[DCTSIZE*0]);
    tmp1 = DEQUANTIZE(inptr[DCTSIZE*2], quantptr[DCTSIZE*2]);
    tmp2 = DEQUANTIZE(inptr[DCTSIZE*4], quantptr[DCTSIZE*4]);
    tmp3 = DEQUANTIZE(inptr[DCTSIZE*6], quantptr[DCTSIZE*6]);

    tmp10 = tmp0 + tmp2; /* phase 3 */
    tmp11 = tmp0 - tmp2;

    tmp13 = tmp1 + tmp3; /* phases 5-3 */
    tmp12 = (tmp1 - tmp3) * ((FAST_FLOAT) 1.414213562) - tmp13; /* 2*c4 */

    tmp0 = tmp10 + tmp13; /* phase 2 */
    tmp3 = tmp10 - tmp13;
    tmp1 = tmp11 + tmp12;
    tmp2 = tmp11 - tmp12;

    /* Odd part */

    tmp4 = DEQUANTIZE(inptr[DCTSIZE*1], quantptr[DCTSIZE*1]);
    tmp5 = DEQUANTIZE(inptr[DCTSIZE*3], quantptr[DCTSIZE*3]);
    tmp6 = DEQUANTIZE(inptr[DCTSIZE*5], quantptr[DCTSIZE*5]);
    tmp7 = DEQUANTIZE(inptr[DCTSIZE*7], quantptr[DCTSIZE*7]);

    z13 = tmp6 + tmp5; /* phase 6 */
    z10 = tmp6 - tmp5;
    z11 = tmp4 + tmp7;
    z12 = tmp4 - tmp7;

    tmp7 = z11 + z13; /* phase 5 */
    tmp11 = (z11 - z13) * ((FAST_FLOAT) 1.414213562); /* 2*c4 */

    z5 = (z10 + z12) * ((FAST_FLOAT) 1.847759065); /* 2*c2 */
    tmp10 = ((FAST_FLOAT) 1.082392200) * z12 - z5; /* 2*(c2-c6) */
    tmp12 = ((FAST_FLOAT) -2.613125930) * z10 + z5; /* -2*(c2+c6) */

    tmp6 = tmp12 - tmp7; /* phase 2 */
    tmp5 = tmp11 - tmp6;
    tmp4 = tmp10 + tmp5;

```

```

wsptr[DCTSIZE*0] = tmp0 + tmp7;
wsptr[DCTSIZE*7] = tmp0 - tmp7;
wsptr[DCTSIZE*1] = tmp1 + tmp6;
wsptr[DCTSIZE*6] = tmp1 - tmp6;
wsptr[DCTSIZE*2] = tmp2 + tmp5;
wsptr[DCTSIZE*5] = tmp2 - tmp5;
wsptr[DCTSIZE*4] = tmp3 + tmp4;
wsptr[DCTSIZE*3] = tmp3 - tmp4;

inptr++;          /* advance pointers to next column */
quantptr++;
wsptr++;
}

/* Pass 2: process rows from work array, store into output array. */
/* Note that we must descale the results by a factor of 8 == 2**3. */

wsptr = workspace;
for (ctr = 0; ctr < DCTSIZE; ctr++) {
    outptr = output_buf[ctr] + output_col;
    /* Rows of zeroes can be exploited in the same way as we did with columns.
     * However, the column calculation has created many nonzero AC terms, so
     * the simplification applies less often (typically 5% to 10% of the time).
     * And testing floats for zero is relatively expensive, so we don't bother.
     */

    /* Even part */

    tmp10 = wsptr[0] + wsptr[4];
    tmp11 = wsptr[0] - wsptr[4];

    tmp13 = wsptr[2] + wsptr[6];
    tmp12 = (wsptr[2] - wsptr[6]) * ((FAST_FLOAT) 1.414213562) - tmp13;

    tmp0 = tmp10 + tmp13;
    tmp3 = tmp10 - tmp13;
    tmp1 = tmp11 + tmp12;
    tmp2 = tmp11 - tmp12;

    /* Odd part */

    z13 = wsptr[5] + wsptr[3];
    z10 = wsptr[5] - wsptr[3];
    z11 = wsptr[1] + wsptr[7];
    z12 = wsptr[1] - wsptr[7];

    tmp7 = z11 + z13;
    tmp11 = (z11 - z13) * ((FAST_FLOAT) 1.414213562);

    z5 = (z10 + z12) * ((FAST_FLOAT) 1.847759065); /* 2*c2 */
    tmp10 = ((FAST_FLOAT) 1.082392200) * z12 - z5; /* 2*(c2-c6) */
    tmp12 = ((FAST_FLOAT) -2.613125930) * z10 + z5; /* -2*(c2+c6) */

    tmp6 = tmp12 - tmp7;
    tmp5 = tmp11 - tmp6;
    tmp4 = tmp10 + tmp5;

    /* Final output stage: scale down by a factor of 8 and range-limit */

    outptr[0] = range_limit[(int) DESCALE((INT32) (tmp0 + tmp7), 3)
        & RANGE_MASK];
    outptr[7] = range_limit[(int) DESCALE((INT32) (tmp0 - tmp7), 3)
        & RANGE_MASK];
    outptr[1] = range_limit[(int) DESCALE((INT32) (tmp1 + tmp6), 3)
        & RANGE_MASK];
    outptr[6] = range_limit[(int) DESCALE((INT32) (tmp1 - tmp6), 3)
        & RANGE_MASK];
    outptr[2] = range_limit[(int) DESCALE((INT32) (tmp2 + tmp5), 3)
        & RANGE_MASK];
    outptr[5] = range_limit[(int) DESCALE((INT32) (tmp2 - tmp5), 3)
        & RANGE_MASK];
    outptr[4] = range_limit[(int) DESCALE((INT32) (tmp3 + tmp4), 3)
        & RANGE_MASK];
    outptr[3] = range_limit[(int) DESCALE((INT32) (tmp3 - tmp4), 3)
        & RANGE_MASK];

    wsptr += DCTSIZE; /* advance pointer to next row */
}
}

```

```
#endif /* DCT_FLOAT_SUPPORTED */
```

```

/*
 * jidctfst.c
 *
 * Copyright (C) 1994-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains a fast, not so accurate integer implementation of the
 * inverse DCT (Discrete Cosine Transform). In the IJG code, this routine
 * must also perform dequantization of the input coefficients.
 *
 * A 2-D IDCT can be done by 1-D IDCT on each column followed by 1-D IDCT
 * on each row (or vice versa, but it's more convenient to emit a row at
 * a time). Direct algorithms are also available, but they are much more
 * complex and seem not to be any faster when reduced to code.
 *
 * This implementation is based on Arai, Agui, and Nakajima's algorithm for
 * scaled DCT. Their original paper (Trans. IEICE E-71(11):1095) is in
 * Japanese, but the algorithm is described in the Pennebaker & Mitchell
 * JPEG textbook (see REFERENCES section in file README). The following code
 * is based directly on figure 4-8 in P&M.
 * While an 8-point DCT cannot be done in less than 11 multiplies, it is
 * possible to arrange the computation so that many of the multiplies are
 * simple scalings of the final outputs. These multiplies can then be
 * folded into the multiplications or divisions by the JPEG quantization
 * table entries. The AA&N method leaves only 5 multiplies and 29 adds
 * to be done in the DCT itself.
 * The primary disadvantage of this method is that with fixed-point math,
 * accuracy is lost due to imprecise representation of the scaled
 * quantization values. The smaller the quantization table entry, the less
 * precise the scaled value, so this implementation does worse with high-
 * quality-setting files than with low-quality ones.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jdct.h" /* Private declarations for DCT subsystem */

#ifdef DCT_IFAST_SUPPORTED

/* This module is specialized to the case DCTSIZE = 8.
 */

#if DCTSIZE != 8
Sorry, this code only copes with 8x8 DCTs. /* deliberate syntax err */
#endif

/* Scaling decisions are generally the same as in the LL&M algorithm;
 * see jidctint.c for more details. However, we choose to descale
 * (right shift) multiplication products as soon as they are formed,
 * rather than carrying additional fractional bits into subsequent additions.
 * This compromises accuracy slightly, but it lets us save a few shifts.
 * More importantly, 16-bit arithmetic is then adequate (for 8-bit samples)
 * everywhere except in the multiplications proper; this saves a good deal
 * of work on 16-bit-int machines.
 *
 * The dequantized coefficients are not integers because the AA&N scaling
 * factors have been incorporated. We represent them scaled up by PASS1_BITS,
 * so that the first and second IDCT rounds have the same input scaling.
 * For 8-bit JSAMPLEs, we choose IFAST_SCALE_BITS = PASS1_BITS so as to
 * avoid a descaling shift; this compromises accuracy rather drastically
 * for small quantization table entries, but it saves a lot of shifts.
 * For 12-bit JSAMPLEs, there's no hope of using 16x16 multiplies anyway,
 * so we use a much larger scaling factor to preserve accuracy.
 *
 * A final compromise is to represent the multiplicative constants to only
 * 8 fractional bits, rather than 13. This saves some shifting work on some
 * machines, and may also reduce the cost of multiplication (since there
 * are fewer one-bits in the constants).
 */

#if BITS_IN_JSAMPLE == 8
#define CONST_BITS 8
#define PASS1_BITS 2
#else
#define CONST_BITS 8

```



```

#define PASS1_BITS 1      /* lose a little precision to avoid overflow */
#endif

/* Some C compilers fail to reduce "FIX(constant)" at compile time, thus
 * causing a lot of useless floating-point operations at run time.
 * To get around this we use the following pre-calculated constants.
 * If you change CONST_BITS you may want to add appropriate values.
 * (With a reasonable C compiler, you can just rely on the FIX() macro...)
 */

#if CONST_BITS == 8
#define FIX_1_082392200 ((INT32) 277) /* FIX(1.082392200) */
#define FIX_1_414213562 ((INT32) 362) /* FIX(1.414213562) */
#define FIX_1_847759065 ((INT32) 473) /* FIX(1.847759065) */
#define FIX_2_613125930 ((INT32) 669) /* FIX(2.613125930) */
#else
#define FIX_1_082392200 FIX(1.082392200)
#define FIX_1_414213562 FIX(1.414213562)
#define FIX_1_847759065 FIX(1.847759065)
#define FIX_2_613125930 FIX(2.613125930)
#endif

/* We can gain a little more speed, with a further compromise in accuracy,
 * by omitting the addition in a descaling shift. This yields an incorrectly
 * rounded result half the time...
 */

#ifndef USE_ACCURATE_ROUNDING
#undef DESCALE
#define DESCALE(x,n) RIGHT_SHIFT(x, n)
#endif

/* Multiply a DCTELEM variable by an INT32 constant, and immediately
 * descale to yield a DCTELEM result.
 */
#define MULTIPLY(var,const) ((DCTELEM) DESCALE((var) * (const), CONST_BITS))

/* Dequantize a coefficient by multiplying it by the multiplier-table
 * entry; produce a DCTELEM result. For 8-bit data a 16x16->16
 * multiplication will do. For 12-bit data, the multiplier table is
 * declared INT32, so a 32-bit multiply will be used.
 */
#if BITS_IN_JSAMPLE == 8
#define DEQUANTIZE(coef,quantval) (((IFAST_MULT_TYPE) (coef)) * (quantval))
#else
#define DEQUANTIZE(coef,quantval) \
    DESCALE((coef)*(quantval), IFAST_SCALE_BITS-PASS1_BITS)
#endif

/* Like DESCALE, but applies to a DCTELEM and produces an int.
 * We assume that int right shift is unsigned if INT32 right shift is.
 */

#ifdef RIGHT_SHIFT_IS_UNSIGNED
#define ISHIFT_TEMPS DCTELEM ishift_temp;
#if BITS_IN_JSAMPLE == 8
#define DCTELEM_BITS 16 /* DCTELEM may be 16 or 32 bits */
#else
#define DCTELEM_BITS 32 /* DCTELEM must be 32 bits */
#endif
#define IRIGHT_SHIFT(x,shft) \
    ((ishift_temp = (x)) < 0 ? \
     (ishift_temp >> (shft)) | ((~((DCTELEM) 0)) << (DCTELEM_BITS-(shft))) : \
     (ishift_temp >> (shft)))
#else
#define ISHIFT_TEMPS
#define IRIGHT_SHIFT(x,shft) ((x) >> (shft))
#endif

#ifdef USE_ACCURATE_ROUNDING
#define IDESCALE(x,n) ((int) IRIGHT_SHIFT((x) + (1 << ((n)-1)), n))
#else
#define IDESCALE(x,n) ((int) IRIGHT_SHIFT(x, n))
#endif

```

```

/*
 * Perform dequantization and inverse DCT on one block of coefficients.
 */

GLOBAL(void)
jpeg_idct_ifast (j_decompress_ptr cinfo, jpeg_component_info * compptr,
                 JCOEFPTR coef_block,
                 JSAMPARRAY output_buf, JDIMENSION output_col)
{
    DCTELEM tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;
    DCTELEM tmp10, tmp11, tmp12, tmp13;
    DCTELEM z5, z10, z11, z12, z13;
    JCOEFPTR inptr;
    IFAST_MULT_TYPE * quantptr;
    int * wsptr;
    JSAMPROW outptr;
    JSAMPLE *range_limit = IDCT_range_limit(cinfo);
    int ctr;
    int workspace[DCTSIZE2]; /* buffers data between passes */
    SHIFT_TEMPS /* for DESCALE */
    ISHIFT_TEMPS /* for IDESCALE */

    /* Pass 1: process columns from input, store into work array. */

    inptr = coef_block;
    quantptr = (IFAST_MULT_TYPE *) compptr->dct_table;
    wsptr = workspace;
    for (ctr = DCTSIZE; ctr > 0; ctr--) {
        /* Due to quantization, we will usually find that many of the input
         * coefficients are zero, especially the AC terms. We can exploit this
         * by short-circuiting the IDCT calculation for any column in which all
         * the AC terms are zero. In that case each output is equal to the
         * DC coefficient (with scale factor as needed).
         * With typical images and quantization tables, half or more of the
         * column DCT calculations can be simplified this way.
         */

        if (inptr[DCTSIZE*1] == 0 && inptr[DCTSIZE*2] == 0 &&
            inptr[DCTSIZE*3] == 0 && inptr[DCTSIZE*4] == 0 &&
            inptr[DCTSIZE*5] == 0 && inptr[DCTSIZE*6] == 0 &&
            inptr[DCTSIZE*7] == 0) {
            /* AC terms all zero */
            int dval = (int) DEQUANTIZE(inptr[DCTSIZE*0], quantptr[DCTSIZE*0]);

            wsptr[DCTSIZE*0] = dval;
            wsptr[DCTSIZE*1] = dval;
            wsptr[DCTSIZE*2] = dval;
            wsptr[DCTSIZE*3] = dval;
            wsptr[DCTSIZE*4] = dval;
            wsptr[DCTSIZE*5] = dval;
            wsptr[DCTSIZE*6] = dval;
            wsptr[DCTSIZE*7] = dval;

            inptr++; /* advance pointers to next column */
            quantptr++;
            wsptr++;
            continue;
        }

        /* Even part */

        tmp0 = DEQUANTIZE(inptr[DCTSIZE*0], quantptr[DCTSIZE*0]);
        tmp1 = DEQUANTIZE(inptr[DCTSIZE*2], quantptr[DCTSIZE*2]);
        tmp2 = DEQUANTIZE(inptr[DCTSIZE*4], quantptr[DCTSIZE*4]);
        tmp3 = DEQUANTIZE(inptr[DCTSIZE*6], quantptr[DCTSIZE*6]);

        tmp10 = tmp0 + tmp2; /* phase 3 */
        tmp11 = tmp0 - tmp2;

        tmp13 = tmp1 + tmp3; /* phases 5-3 */
        tmp12 = MULTIPLY(tmp1 - tmp3, FIX_1_414213562) - tmp13; /* 2*c4 */

        tmp0 = tmp10 + tmp13; /* phase 2 */
        tmp3 = tmp10 - tmp13;
        tmp1 = tmp11 + tmp12;
        tmp2 = tmp11 - tmp12;

        /* Odd part */
    }
}

```

```

tmp4 = DEQUANTIZE(inp[ptr[DCTSIZE*1]], quantptr[DCTSIZE*1]);
tmp5 = DEQUANTIZE(inp[ptr[DCTSIZE*3]], quantptr[DCTSIZE*3]);
tmp6 = DEQUANTIZE(inp[ptr[DCTSIZE*5]], quantptr[DCTSIZE*5]);
tmp7 = DEQUANTIZE(inp[ptr[DCTSIZE*7]], quantptr[DCTSIZE*7]);

z13 = tmp6 + tmp5;      /* phase 6 */
z10 = tmp6 - tmp5;
z11 = tmp4 + tmp7;
z12 = tmp4 - tmp7;

tmp7 = z11 + z13;      /* phase 5 */
tmp11 = MULTIPLY(z11 - z13, FIX_1_414213562); /* 2*c4 */

z5 = MULTIPLY(z10 + z12, FIX_1_847759065); /* 2*c2 */
tmp10 = MULTIPLY(z12, FIX_1_082392200) - z5; /* 2*(c2-c6) */
tmp12 = MULTIPLY(z10, -FIX_2_613125930) + z5; /* -2*(c2+c6) */

tmp6 = tmp12 - tmp7;    /* phase 2 */
tmp5 = tmp11 - tmp6;
tmp4 = tmp10 + tmp5;

wsptr[DCTSIZE*0] = (int) (tmp0 + tmp7);
wsptr[DCTSIZE*7] = (int) (tmp0 - tmp7);
wsptr[DCTSIZE*1] = (int) (tmp1 + tmp6);
wsptr[DCTSIZE*6] = (int) (tmp1 - tmp6);
wsptr[DCTSIZE*2] = (int) (tmp2 + tmp5);
wsptr[DCTSIZE*5] = (int) (tmp2 - tmp5);
wsptr[DCTSIZE*4] = (int) (tmp3 + tmp4);
wsptr[DCTSIZE*3] = (int) (tmp3 - tmp4);

inp[ptr]++;      /* advance pointers to next column */
quantptr++;
wsptr++;

/* Pass 2: process rows from work array, store into output array. */
/* Note that we must descale the results by a factor of 8 == 2**3, */
/* and also undo the PASS1_BITS scaling. */

wsptr = workspace;
for (ctr = 0; ctr < DCTSIZE; ctr++) {
    outptr = output_buf[ctr] + output_col;
    /* Rows of zeroes can be exploited in the same way as we did with columns.
     * However, the column calculation has created many nonzero AC terms, so
     * the simplification applies less often (typically 5% to 10% of the time).
     * On machines with very fast multiplication, it's possible that the
     * test takes more time than it's worth. In that case this section
     * may be commented out.
     */
    #ifndef NO_ZERO_ROW_TEST
    if (wsptr[1] == 0 && wsptr[2] == 0 && wsptr[3] == 0 && wsptr[4] == 0 &&
        wsptr[5] == 0 && wsptr[6] == 0 && wsptr[7] == 0) {
        /* AC terms all zero */
        JSAMPLE dcv[0] = range_limit[IDESCALE(wsptr[0], PASS1_BITS+3)
            & RANGE_MASK];

        outptr[0] = dcv[0];
        outptr[1] = dcv[0];
        outptr[2] = dcv[0];
        outptr[3] = dcv[0];
        outptr[4] = dcv[0];
        outptr[5] = dcv[0];
        outptr[6] = dcv[0];
        outptr[7] = dcv[0];

        wsptr += DCTSIZE;      /* advance pointer to next row */
        continue;
    }
}
#endif

/* Even part */

tmp10 = ((DCTELEM) wsptr[0] + (DCTELEM) wsptr[4]);
tmp11 = ((DCTELEM) wsptr[0] - (DCTELEM) wsptr[4]);

tmp13 = ((DCTELEM) wsptr[2] + (DCTELEM) wsptr[6]);
tmp12 = MULTIPLY((DCTELEM) wsptr[2] - (DCTELEM) wsptr[6], FIX_1_414213562)
    - tmp13;

```

```

tmp0 = tmp10 + tmp13;
tmp3 = tmp10 - tmp13;
tmp1 = tmp11 + tmp12;
tmp2 = tmp11 - tmp12;

/* Odd part */

z13 = (DCTELEM) wsptr[5] + (DCTELEM) wsptr[3];
z10 = (DCTELEM) wsptr[5] - (DCTELEM) wsptr[3];
z11 = (DCTELEM) wsptr[1] + (DCTELEM) wsptr[7];
z12 = (DCTELEM) wsptr[1] - (DCTELEM) wsptr[7];

tmp7 = z11 + z13;          /* phase 5 */
tmp11 = MULTIPLY(z11 - z13, FIX_1_414213562); /* 2*c4 */

z5 = MULTIPLY(z10 + z12, FIX_1_847759065); /* 2*c2 */
tmp10 = MULTIPLY(z12, FIX_1_082392200) - z5; /* 2*(c2-c6) */
tmp12 = MULTIPLY(z10, - FIX_2_613125930) + z5; /* -2*(c2+c6) */

tmp6 = tmp12 - tmp7;      /* phase 2 */
tmp5 = tmp11 - tmp6;
tmp4 = tmp10 + tmp5;

/* Final output stage: scale down by a factor of 8 and range-limit */

outptr[0] = range_limit[IDESCALE(tmp0 + tmp7, PASS1_BITS+3)
    & RANGE_MASK];
outptr[7] = range_limit[IDESCALE(tmp0 - tmp7, PASS1_BITS+3)
    & RANGE_MASK];
outptr[1] = range_limit[IDESCALE(tmp1 + tmp6, PASS1_BITS+3)
    & RANGE_MASK];
outptr[6] = range_limit[IDESCALE(tmp1 - tmp6, PASS1_BITS+3)
    & RANGE_MASK];
outptr[2] = range_limit[IDESCALE(tmp2 + tmp5, PASS1_BITS+3)
    & RANGE_MASK];
outptr[5] = range_limit[IDESCALE(tmp2 - tmp5, PASS1_BITS+3)
    & RANGE_MASK];
outptr[4] = range_limit[IDESCALE(tmp3 + tmp4, PASS1_BITS+3)
    & RANGE_MASK];
outptr[3] = range_limit[IDESCALE(tmp3 - tmp4, PASS1_BITS+3)
    & RANGE_MASK];

wsptr += DCTSIZE;          /* advance pointer to next row */

#endif /* DCT_IFAST_SUPPORTED */

```

```

/*
 * jidctint.c
 *
 * Copyright (C) 1991-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains a slow-but-accurate integer implementation of the
 * inverse DCT (Discrete Cosine Transform). In the IJG code, this routine
 * must also perform dequantization of the input coefficients.
 *
 * A 2-D IDCT can be done by 1-D IDCT on each column followed by 1-D IDCT
 * on each row (or vice versa, but it's more convenient to emit a row at
 * a time). Direct algorithms are also available, but they are much more
 * complex and seem not to be any faster when reduced to code.
 *
 * This implementation is based on an algorithm described in
 *   C. Loeffler, A. Ligtenberg and G. Moschytz, "Practical Fast 1-D DCT
 *   Algorithms with 11 Multiplications", Proc. Int'l. Conf. on Acoustics,
 *   Speech, and Signal Processing 1989 (ICASSP '89), pp. 988-991.
 * The primary algorithm described there uses 11 multiplies and 29 adds.
 * We use their alternate method with 12 multiplies and 32 adds.
 * The advantage of this method is that no data path contains more than one
 * multiplication; this allows a very simple and accurate implementation in
 * scaled fixed-point arithmetic, with a minimal number of shifts.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jdct.h"          /* Private declarations for DCT subsystem */

#ifdef DCT_ISLOW_SUPPORTED

/* This module is specialized to the case DCTSIZE = 8.
 */

#if DCTSIZE != 8
  Sorry, this code only copes with 8x8 DCTs. /* deliberate syntax err */
#endif

/*
 * The poop on this scaling stuff is as follows:
 *
 * Each 1-D IDCT step produces outputs which are a factor of sqrt(N)
 * larger than the true IDCT outputs. The final outputs are therefore
 * a factor of N larger than desired; since N=8 this can be cured by
 * a simple right shift at the end of the algorithm. The advantage of
 * this arrangement is that we save two multiplications per 1-D IDCT,
 * because the y0 and y4 inputs need not be divided by sqrt(N).
 *
 * We have to do addition and subtraction of the integer inputs, which
 * is no problem, and multiplication by fractional constants, which is
 * a problem to do in integer arithmetic. We multiply all the constants
 * by CONST_SCALE and convert them to integer constants (thus retaining
 * CONST_BITS bits of precision in the constants). After doing a
 * multiplication we have to divide the product by CONST_SCALE, with proper
 * rounding, to produce the correct output. This division can be done
 * cheaply as a right shift of CONST_BITS bits. We postpone shifting
 * as long as possible so that partial sums can be added together with
 * full fractional precision.
 *
 * The outputs of the first pass are scaled up by PASS1_BITS bits so that
 * they are represented to better-than-integral precision. These outputs
 * require BITS_IN_JSAMPLE + PASS1_BITS + 3 bits; this fits in a 16-bit word
 * with the recommended scaling. (To scale up 12-bit sample data further, an
 * intermediate INT32 array would be needed.)
 *
 * To avoid overflow of the 32-bit intermediate results in pass 2, we must
 * have BITS_IN_JSAMPLE + CONST_BITS + PASS1_BITS <= 26. Error analysis
 * shows that the values given below are the most effective.
 */

#if BITS_IN_JSAMPLE == 8
#define CONST_BITS 13
#define PASS1_BITS 2
#else

```

```

#define CONST_BITS 13
#define PASS1_BITS 1      /* lose a little precision to avoid overflow */
#endif

/* Some C compilers fail to reduce "FIX(constant)" at compile time, thus
 * causing a lot of useless floating-point operations at run time.
 * To get around this we use the following pre-calculated constants.
 * If you change CONST_BITS you may want to add appropriate values.
 * (With a reasonable C compiler, you can just rely on the FIX() macro...)
 */

#if CONST_BITS == 13
#define FIX_0_298631336 ((INT32) 2446) /* FIX(0.298631336) */
#define FIX_0_390180644 ((INT32) 3196) /* FIX(0.390180644) */
#define FIX_0_541196100 ((INT32) 4433) /* FIX(0.541196100) */
#define FIX_0_765366865 ((INT32) 6270) /* FIX(0.765366865) */
#define FIX_0_899976223 ((INT32) 7373) /* FIX(0.899976223) */
#define FIX_1_175875602 ((INT32) 9633) /* FIX(1.175875602) */
#define FIX_1_501321110 ((INT32) 12299) /* FIX(1.501321110) */
#define FIX_1_847759065 ((INT32) 15137) /* FIX(1.847759065) */
#define FIX_1_961570560 ((INT32) 16069) /* FIX(1.961570560) */
#define FIX_2_053119869 ((INT32) 16819) /* FIX(2.053119869) */
#define FIX_2_562915447 ((INT32) 20995) /* FIX(2.562915447) */
#define FIX_3_072711026 ((INT32) 25172) /* FIX(3.072711026) */
#else
#define FIX_0_298631336 FIX(0.298631336)
#define FIX_0_390180644 FIX(0.390180644)
#define FIX_0_541196100 FIX(0.541196100)
#define FIX_0_765366865 FIX(0.765366865)
#define FIX_0_899976223 FIX(0.899976223)
#define FIX_1_175875602 FIX(1.175875602)
#define FIX_1_501321110 FIX(1.501321110)
#define FIX_1_847759065 FIX(1.847759065)
#define FIX_1_961570560 FIX(1.961570560)
#define FIX_2_053119869 FIX(2.053119869)
#define FIX_2_562915447 FIX(2.562915447)
#define FIX_3_072711026 FIX(3.072711026)
#endif

/*
 * Multiply an INT32 variable by an INT32 constant to yield an INT32 result.
 * For 8-bit samples with the recommended scaling, all the variable
 * and constant values involved are no more than 16 bits wide, so a
 * 16x16->32 bit multiply can be used instead of a full 32x32 multiply.
 * For 12-bit samples, a full 32-bit multiplication will be needed.
 */

#if BITS_IN_JSAMPLE == 8
#define MULTIPLY(var,const) MULTIPLY16C16(var,const)
#else
#define MULTIPLY(var,const) ((var) * (const))
#endif

/* Dequantize a coefficient by multiplying it by the multiplier-table
 * entry; produce an int result. In this module, both inputs and result
 * are 16 bits or less, so either int or short multiply will work.
 */

#define DEQUANTIZE(coef,quantval) (((ISLOW_MULT_TYPE) (coef)) * (quantval))

/*
 * Perform dequantization and inverse DCT on one block of coefficients.
 */

GLOBAL(void)
jpeg_idct_islow (j_decompress_ptr cinfo, jpeg_component_info * comp_ptr,
                 JCOEFPTR coef_block,
                 JSAMPARRAY output_buf, JDIMENSION output_col)
{
    INT32 tmp0, tmp1, tmp2, tmp3;
    INT32 tmp10, tmp11, tmp12, tmp13;
    INT32 z1, z2, z3, z4, z5;
    JCOEFPTR in_ptr;
    ISLOW_MULT_TYPE * quant_ptr;
    int * wsptr;
    JSAMPROW out_ptr;
    JSAMPLE *range_limit = IDCT_range_limit(cinfo);
    int ctr;

```

```

int workspace[DCTSIZE2]; /* buffers data between passes */
SHIFT_TEMPS

/* Pass 1: process columns from input, store into work array. */
/* Note results are scaled up by sqrt(8) compared to a true IDCT; */
/* furthermore, we scale the results by 2**PASS1_BITS. */

inptr = coef_block;
quantptr = (ISLOW_MULT_TYPE *) compptr->dct_table;
wsptr = workspace;
for (ctr = DCTSIZE; ctr > 0; ctr--) {
    /* Due to quantization, we will usually find that many of the input
     * coefficients are zero, especially the AC terms. We can exploit this
     * by short-circuiting the IDCT calculation for any column in which all
     * the AC terms are zero. In that case each output is equal to the
     * DC coefficient (with scale factor as needed).
     * With typical images and quantization tables, half or more of the
     * column DCT calculations can be simplified this way.
     */

    if (inptr[DCTSIZE*1] == 0 && inptr[DCTSIZE*2] == 0 &&
        inptr[DCTSIZE*3] == 0 && inptr[DCTSIZE*4] == 0 &&
        inptr[DCTSIZE*5] == 0 && inptr[DCTSIZE*6] == 0 &&
        inptr[DCTSIZE*7] == 0) {
        /* AC terms all zero */
        int dval = DEQUANTIZE(inptr[DCTSIZE*0], quantptr[DCTSIZE*0]) << PASS1_BITS;

        wsptr[DCTSIZE*0] = dval;
        wsptr[DCTSIZE*1] = dval;
        wsptr[DCTSIZE*2] = dval;
        wsptr[DCTSIZE*3] = dval;
        wsptr[DCTSIZE*4] = dval;
        wsptr[DCTSIZE*5] = dval;
        wsptr[DCTSIZE*6] = dval;
        wsptr[DCTSIZE*7] = dval;

        inptr++;          /* advance pointers to next column */
        quantptr++;
        wsptr++;
        continue;
    }

    /* Even part: reverse the even part of the forward DCT. */
    /* The rotator is sqrt(2)*c(-6). */

    z2 = DEQUANTIZE(inptr[DCTSIZE*2], quantptr[DCTSIZE*2]);
    z3 = DEQUANTIZE(inptr[DCTSIZE*6], quantptr[DCTSIZE*6]);

    z1 = MULTIPLY(z2 + z3, FIX_0_541196100);
    tmp2 = z1 + MULTIPLY(z3, -FIX_1_847759065);
    tmp3 = z1 + MULTIPLY(z2, FIX_0_765366865);

    z2 = DEQUANTIZE(inptr[DCTSIZE*0], quantptr[DCTSIZE*0]);
    z3 = DEQUANTIZE(inptr[DCTSIZE*4], quantptr[DCTSIZE*4]);

    tmp0 = (z2 + z3) << CONST_BITS;
    tmp1 = (z2 - z3) << CONST_BITS;

    tmp10 = tmp0 + tmp3;
    tmp13 = tmp0 - tmp3;
    tmp11 = tmp1 + tmp2;
    tmp12 = tmp1 - tmp2;

    /* Odd part per figure 8; the matrix is unitary and hence its
     * transpose is its inverse. i0..i3 are y7,y5,y3,y1 respectively.
     */

    tmp0 = DEQUANTIZE(inptr[DCTSIZE*7], quantptr[DCTSIZE*7]);
    tmp1 = DEQUANTIZE(inptr[DCTSIZE*5], quantptr[DCTSIZE*5]);
    tmp2 = DEQUANTIZE(inptr[DCTSIZE*3], quantptr[DCTSIZE*3]);
    tmp3 = DEQUANTIZE(inptr[DCTSIZE*1], quantptr[DCTSIZE*1]);

    z1 = tmp0 + tmp3;
    z2 = tmp1 + tmp2;
    z3 = tmp0 + tmp2;
    z4 = tmp1 + tmp3;
    z5 = MULTIPLY(z3 + z4, FIX_1_175875602); /* sqrt(2) * c3 */

    tmp0 = MULTIPLY(tmp0, FIX_0_298631336); /* sqrt(2) * (-c1+c3+c5-c7) */
    tmp1 = MULTIPLY(tmp1, FIX_2_053119869); /* sqrt(2) * ( c1+c3-c5+c7) */

```

```

tmp2 = MULTIPLY(tmp2, FIX_3_072711026); /* sqrt(2) * ( c1+c3+c5-c7) */
tmp3 = MULTIPLY(tmp3, FIX_1_501321110); /* sqrt(2) * ( c1+c3-c5-c7) */
z1 = MULTIPLY(z1, - FIX_0_899976223); /* sqrt(2) * (c7-c3) */
z2 = MULTIPLY(z2, - FIX_2_562915447); /* sqrt(2) * (-c1-c3) */
z3 = MULTIPLY(z3, - FIX_1_961570560); /* sqrt(2) * (-c3-c5) */
z4 = MULTIPLY(z4, - FIX_0_390180644); /* sqrt(2) * (c5-c3) */

z3 += z5;
z4 += z5;

tmp0 += z1 + z3;
tmp1 += z2 + z4;
tmp2 += z2 + z3;
tmp3 += z1 + z4;

/* Final output stage: inputs are tmp10..tmp13, tmp0..tmp3 */
wsptr[DCTSIZE*0] = (int) DESCALE(tmp10 + tmp3, CONST_BITS-PASS1_BITS);
wsptr[DCTSIZE*7] = (int) DESCALE(tmp10 - tmp3, CONST_BITS-PASS1_BITS);
wsptr[DCTSIZE*1] = (int) DESCALE(tmp11 + tmp2, CONST_BITS-PASS1_BITS);
wsptr[DCTSIZE*6] = (int) DESCALE(tmp11 - tmp2, CONST_BITS-PASS1_BITS);
wsptr[DCTSIZE*2] = (int) DESCALE(tmp12 + tmp1, CONST_BITS-PASS1_BITS);
wsptr[DCTSIZE*5] = (int) DESCALE(tmp12 - tmp1, CONST_BITS-PASS1_BITS);
wsptr[DCTSIZE*3] = (int) DESCALE(tmp13 + tmp0, CONST_BITS-PASS1_BITS);
wsptr[DCTSIZE*4] = (int) DESCALE(tmp13 - tmp0, CONST_BITS-PASS1_BITS);

inptr++; /* advance pointers to next column */
quantptr++;
wsptr++;
}

/* Pass 2: process rows from work array, store into output array. */
/* Note that we must descale the results by a factor of 8 == 2**3, */
/* and also undo the PASS1_BITS scaling. */
wsptr = workspace;
for (ctr = 0; ctr < DCTSIZE; ctr++) {
    outptr = output_buf[ctr] + output_col;
    /* Rows of zeroes can be exploited in the same way as we did with columns.
     * However, the column calculation has created many nonzero AC terms, so
     * the simplification applies less often (typically 5% to 10% of the time).
     * On machines with very fast multiplication, it's possible that the
     * test takes more time than it's worth. In that case this section
     * may be commented out.
     */
    #ifndef NO_ZERO_ROW_TEST
    if (wsptr[1] == 0 && wsptr[2] == 0 && wsptr[3] == 0 && wsptr[4] == 0 &&
        wsptr[5] == 0 && wsptr[6] == 0 && wsptr[7] == 0) {
        /* AC terms all zero */
        JSAMPLE dcvall = range_limit[(int) DESCALE((INT32) wsptr[0], PASS1_BITS+3)
            & RANGE_MASK];
        outptr[0] = dcvall;
        outptr[1] = dcvall;
        outptr[2] = dcvall;
        outptr[3] = dcvall;
        outptr[4] = dcvall;
        outptr[5] = dcvall;
        outptr[6] = dcvall;
        outptr[7] = dcvall;

        wsptr += DCTSIZE; /* advance pointer to next row */
        continue;
    }
    #endif

    /* Even part: reverse the even part of the forward DCT. */
    /* The rotator is sqrt(2)*c(-6). */

    z2 = (INT32) wsptr[2];
    z3 = (INT32) wsptr[6];

    z1 = MULTIPLY(z2 + z3, FIX_0_541196100);
    tmp2 = z1 + MULTIPLY(z3, - FIX_1_847759065);
    tmp3 = z1 + MULTIPLY(z2, FIX_0_765366865);

    tmp0 = ((INT32) wsptr[0] + (INT32) wsptr[4]) << CONST_BITS;
    tmp1 = ((INT32) wsptr[0] - (INT32) wsptr[4]) << CONST_BITS;

```



```

tmp10 = tmp0 + tmp3;
tmp13 = tmp0 - tmp3;
tmp11 = tmp1 + tmp2;
tmp12 = tmp1 - tmp2;

/* Odd part per figure 8; the matrix is unitary and hence its
 * transpose is its inverse. i0..i3 are y7,y5,y3,y1 respectively.
 */

tmp0 = (INT32) wsptr[7];
tmp1 = (INT32) wsptr[5];
tmp2 = (INT32) wsptr[3];
tmp3 = (INT32) wsptr[1];

z1 = tmp0 + tmp3;
z2 = tmp1 + tmp2;
z3 = tmp0 + tmp2;
z4 = tmp1 + tmp3;
z5 = MULTIPLY(z3 + z4, FIX_1_175875602); /* sqrt(2) * c3 */

tmp0 = MULTIPLY(tmp0, FIX_0_298631336); /* sqrt(2) * (-c1+c3+c5-c7) */
tmp1 = MULTIPLY(tmp1, FIX_2_053119869); /* sqrt(2) * ( c1+c3-c5+c7) */
tmp2 = MULTIPLY(tmp2, FIX_3_072711026); /* sqrt(2) * ( c1+c3+c5-c7) */
tmp3 = MULTIPLY(tmp3, FIX_1_501321110); /* sqrt(2) * ( c1+c3-c5-c7) */
z1 = MULTIPLY(z1, - FIX_0_899976223); /* sqrt(2) * (c7-c3) */
z2 = MULTIPLY(z2, - FIX_2_562915447); /* sqrt(2) * (-c1-c3) */
z3 = MULTIPLY(z3, - FIX_1_961570560); /* sqrt(2) * (-c3-c5) */
z4 = MULTIPLY(z4, - FIX_0_390180644); /* sqrt(2) * (c5-c3) */

z3 += z5;
z4 += z5;

tmp0 += z1 + z3;
tmp1 += z2 + z4;
tmp2 += z2 + z3;
tmp3 += z1 + z4;

/* Final output stage: inputs are tmp10..tmp13, tmp0..tmp3 */
outptr[0] = range_limit[(int) DESCALE(tmp10 + tmp3,
CONST_BITS+PASS1_BITS+3)
& RANGE_MASK];
outptr[7] = range_limit[(int) DESCALE(tmp10 - tmp3,
CONST_BITS+PASS1_BITS+3)
& RANGE_MASK];
outptr[1] = range_limit[(int) DESCALE(tmp11 + tmp2,
CONST_BITS+PASS1_BITS+3)
& RANGE_MASK];
outptr[6] = range_limit[(int) DESCALE(tmp11 - tmp2,
CONST_BITS+PASS1_BITS+3)
& RANGE_MASK];
outptr[2] = range_limit[(int) DESCALE(tmp12 + tmp1,
CONST_BITS+PASS1_BITS+3)
& RANGE_MASK];
outptr[5] = range_limit[(int) DESCALE(tmp12 - tmp1,
CONST_BITS+PASS1_BITS+3)
& RANGE_MASK];
outptr[3] = range_limit[(int) DESCALE(tmp13 + tmp0,
CONST_BITS+PASS1_BITS+3)
& RANGE_MASK];
outptr[4] = range_limit[(int) DESCALE(tmp13 - tmp0,
CONST_BITS+PASS1_BITS+3)
& RANGE_MASK];

wsptr += DCTSIZE; /* advance pointer to next row */
}
}
#endif /* DCT_ISLOW_SUPPORTED */

```

```

/*
 * jidctred.c
 *
 * Copyright (C) 1994-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains inverse-DCT routines that produce reduced-size output:
 * either 4x4, 2x2, or 1x1 pixels from an 8x8 DCT block.
 *
 * The implementation is based on the Loeffler, Ligtenberg and Moschytz (LL&M)
 * algorithm used in jidctint.c. We simply replace each 8-to-8 1-D IDCT step
 * with an 8-to-4 step that produces the four averages of two adjacent outputs
 * (or an 8-to-2 step producing two averages of four outputs, for 2x2 output).
 * These steps were derived by computing the corresponding values at the end
 * of the normal LL&M code, then simplifying as much as possible.
 *
 * 1x1 is trivial: just take the DC coefficient divided by 8.
 *
 * See jidctint.c for additional comments.
 */

```

```

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jdct.h" /* Private declarations for DCT subsystem */

```

```

#ifdef IDCT_SCALING_SUPPORTED

```

```

/*
 * This module is specialized to the case DCTSIZE = 8.
 */
#if DCTSIZE != 8
  Sorry, this code only copes with 8x8 DCTs. /* deliberate syntax err */
#endif

/* Scaling is the same as in jidctint.c. */
#if BITS_IN_JSAMPLE == 8
#define CONST_BITS 13
#define PASS1_BITS 2
#else
#define CONST_BITS 13
#define PASS1_BITS 1 /* lose a little precision to avoid overflow */
#endif

/* Some C compilers fail to reduce "FIX(constant)" at compile time, thus
 * causing a lot of useless floating-point operations at run time.
 * To get around this we use the following pre-calculated constants.
 * If you change CONST_BITS you may want to add appropriate values.
 * (With a reasonable C compiler, you can just rely on the FIX() macro...)
 */

```

```

#if CONST_BITS == 13
#define FIX_0_211164243 ((INT32) 1730) /* FIX(0.211164243) */
#define FIX_0_509795579 ((INT32) 4176) /* FIX(0.509795579) */
#define FIX_0_601344887 ((INT32) 4926) /* FIX(0.601344887) */
#define FIX_0_720959822 ((INT32) 5906) /* FIX(0.720959822) */
#define FIX_0_765366865 ((INT32) 6270) /* FIX(0.765366865) */
#define FIX_0_850430095 ((INT32) 6967) /* FIX(0.850430095) */
#define FIX_0_899976223 ((INT32) 7373) /* FIX(0.899976223) */
#define FIX_1_061594337 ((INT32) 8697) /* FIX(1.061594337) */
#define FIX_1_272758580 ((INT32) 10426) /* FIX(1.272758580) */
#define FIX_1_451774981 ((INT32) 11893) /* FIX(1.451774981) */
#define FIX_1_847759065 ((INT32) 15137) /* FIX(1.847759065) */
#define FIX_2_172734803 ((INT32) 17799) /* FIX(2.172734803) */
#define FIX_2_562915447 ((INT32) 20995) /* FIX(2.562915447) */
#define FIX_3_624509785 ((INT32) 29692) /* FIX(3.624509785) */
#else
#define FIX_0_211164243 FIX(0.211164243)
#define FIX_0_509795579 FIX(0.509795579)
#define FIX_0_601344887 FIX(0.601344887)
#define FIX_0_720959822 FIX(0.720959822)
#define FIX_0_765366865 FIX(0.765366865)
#define FIX_0_850430095 FIX(0.850430095)
#define FIX_0_899976223 FIX(0.899976223)
#define FIX_1_061594337 FIX(1.061594337)

```

```

#define FIX_1_272758580    FIX(1.272758580)
#define FIX_1_451774981    FIX(1.451774981)
#define FIX_1_847759065    FIX(1.847759065)
#define FIX_2_172734803    FIX(2.172734803)
#define FIX_2_562915447    FIX(2.562915447)
#define FIX_3_624509785    FIX(3.624509785)
#endif

/* Multiply an INT32 variable by an INT32 constant to yield an INT32 result.
 * For 8-bit samples with the recommended scaling, all the variable
 * and constant values involved are no more than 16 bits wide, so a
 * 16x16->32 bit multiply can be used instead of a full 32x32 multiply.
 * For 12-bit samples, a full 32-bit multiplication will be needed.
 */

#if BITS_IN_JSAMPLE == 8
#define MULTIPLY(var,const)  MULTIPLY16C16(var,const)
#else
#define MULTIPLY(var,const)  ((var) * (const))
#endif

/* Dequantize a coefficient by multiplying it by the multiplier-table
 * entry; produce an int result.  In this module, both inputs and result
 * are 16 bits or less, so either int or short multiply will work.
 */

#define DEQUANTIZE(coef,quantval)  (((ISLOW_MULT_TYPE) (coef)) * (quantval))

/*
 * Perform dequantization and inverse DCT on one block of coefficients,
 * producing a reduced-size 4x4 output block.
 */

GLOBAL(void)
jpeg_idct_4x4 (j_decompress_ptr cinfo, jpeg_component_info * comp_ptr,
               JCOEFPTR coef_block,
               JSAMPARRAY output_buf, JDIMENSION output_col)
{
    INT32 tmp0, tmp2, tmp10, tmp12;
    INT32 z1, z2, z3, z4;
    JCOEFPTR inptr;
    ISLOW_MULT_TYPE * quantptr;
    int * wsptr;
    JSAMPROW outptr;
    JSAMPLE *range_limit = IDCT_range_limit(cinfo);
    int ctr;
    int workspace[DCTSIZE*4]; /* buffers data between passes */
    SHIFT_TEMPS

    /* Pass 1: process columns from input, store into work array. */

    inptr = coef_block;
    quantptr = (ISLOW_MULT_TYPE *) comp_ptr->dct_table;
    wsptr = workspace;
    for (ctr = DCTSIZE; ctr > 0; inptr++, quantptr++, wsptr++, ctr--) {
        /* Don't bother to process column 4, because second pass won't use it */
        if (ctr == DCTSIZE-4)
            continue;
        if (inptr[DCTSIZE*1] == 0 && inptr[DCTSIZE*2] == 0 &&
            inptr[DCTSIZE*3] == 0 && inptr[DCTSIZE*5] == 0 &&
            inptr[DCTSIZE*6] == 0 && inptr[DCTSIZE*7] == 0) {
            /* AC terms all zero; we need not examine term 4 for 4x4 output */
            int dcv = DEQUANTIZE(inptr[DCTSIZE*0], quantptr[DCTSIZE*0]) << PASS1_BITS;

            wsptr[DCTSIZE*0] = dcv;
            wsptr[DCTSIZE*1] = dcv;
            wsptr[DCTSIZE*2] = dcv;
            wsptr[DCTSIZE*3] = dcv;

            continue;
        }

        /* Even part */

        tmp0 = DEQUANTIZE(inptr[DCTSIZE*0], quantptr[DCTSIZE*0]);
        tmp0 <<= (CONST_BITS+1);

```

```

z2 = DEQUANTIZE(inptr[DCTSIZE*2], quantptr[DCTSIZE*2]);
z3 = DEQUANTIZE(inptr[DCTSIZE*6], quantptr[DCTSIZE*6]);

tmp2 = MULTIPLY(z2, FIX_1_847759065) + MULTIPLY(z3, - FIX_0_765366865);

tmp10 = tmp0 + tmp2;
tmp12 = tmp0 - tmp2;

/* Odd part */

z1 = DEQUANTIZE(inptr[DCTSIZE*7], quantptr[DCTSIZE*7]);
z2 = DEQUANTIZE(inptr[DCTSIZE*5], quantptr[DCTSIZE*5]);
z3 = DEQUANTIZE(inptr[DCTSIZE*3], quantptr[DCTSIZE*3]);
z4 = DEQUANTIZE(inptr[DCTSIZE*1], quantptr[DCTSIZE*1]);

tmp0 = MULTIPLY(z1, - FIX_0_211164243) /* sqrt(2) * (c3-c1) */
+ MULTIPLY(z2, FIX_1_451774981) /* sqrt(2) * (c3+c7) */
+ MULTIPLY(z3, - FIX_0_172734803) /* sqrt(2) * (-c1-c5) */
+ MULTIPLY(z4, FIX_1_061594337) /* sqrt(2) * (c5+c7) */

tmp2 = MULTIPLY(z1, - FIX_0_509795579) /* sqrt(2) * (c7-c5) */
+ MULTIPLY(z2, - FIX_0_601344887) /* sqrt(2) * (c5-c1) */
+ MULTIPLY(z3, FIX_0_899976223) /* sqrt(2) * (c3-c7) */
+ MULTIPLY(z4, FIX_2_562915447) /* sqrt(2) * (c1+c3) */

/* Final output stage */

wsptr[DCTSIZE*0] = (int) DESCALE(tmp10 + tmp2, CONST_BITS-PASS1_BITS+1);
wsptr[DCTSIZE*3] = (int) DESCALE(tmp10 - tmp2, CONST_BITS-PASS1_BITS+1);
wsptr[DCTSIZE*1] = (int) DESCALE(tmp12 + tmp0, CONST_BITS-PASS1_BITS+1);
wsptr[DCTSIZE*2] = (int) DESCALE(tmp12 - tmp0, CONST_BITS-PASS1_BITS+1);

/* Pass 2: process 4 rows from work array, store into output array. */
wsptr = workspace;
for (ctr = 0; ctr < 4; ctr++) {
    outptr = output_buf[ctr] + output_col;
    /* It's not clear whether a zero row test is worthwhile here ... */

#ifdef NO_ZERO_ROW_TEST
    if (wsptr[1] == 0 && wsptr[2] == 0 && wsptr[3] == 0 &&
        wsptr[5] == 0 && wsptr[6] == 0 && wsptr[7] == 0) {
        /* AC terms all zero */
        JSAMPLE dcvall = range_limit[(int) DESCALE((INT32) wsptr[0], PASS1_BITS+3)
            & RANGE_MASK];

        outptr[0] = dcvall;
        outptr[1] = dcvall;
        outptr[2] = dcvall;
        outptr[3] = dcvall;

        wsptr += DCTSIZE; /* advance pointer to next row */
        continue;
    }
#endif
}

/* Even part */

tmp0 = ((INT32) wsptr[0]) << (CONST_BITS+1);

tmp2 = MULTIPLY((INT32) wsptr[2], FIX_1_847759065)
+ MULTIPLY((INT32) wsptr[6], - FIX_0_765366865);

tmp10 = tmp0 + tmp2;
tmp12 = tmp0 - tmp2;

/* Odd part */

z1 = (INT32) wsptr[7];
z2 = (INT32) wsptr[5];
z3 = (INT32) wsptr[3];
z4 = (INT32) wsptr[1];

tmp0 = MULTIPLY(z1, - FIX_0_211164243) /* sqrt(2) * (c3-c1) */
+ MULTIPLY(z2, FIX_1_451774981) /* sqrt(2) * (c3+c7) */
+ MULTIPLY(z3, - FIX_0_172734803) /* sqrt(2) * (-c1-c5) */
+ MULTIPLY(z4, FIX_1_061594337) /* sqrt(2) * (c5+c7) */

tmp2 = MULTIPLY(z1, - FIX_0_509795579) /* sqrt(2) * (c7-c5) */

```

```

+ MULTIPLY(z2, - FIX_0_601344887) /* sqrt(2) * (c5-c1) */
+ MULTIPLY(z3, FIX_0_899976223) /* sqrt(2) * (c3-c7) */
+ MULTIPLY(z4, FIX_2_562915447); /* sqrt(2) * (c1+c3) */

/* Final output stage */

outptr[0] = range_limit[(int) DESCALE(tmp10 + tmp2,
CONST_BITS+PASS1_BITS+3+1)
& RANGE_MASK];
outptr[3] = range_limit[(int) DESCALE(tmp10 - tmp2,
CONST_BITS+PASS1_BITS+3+1)
& RANGE_MASK];
outptr[1] = range_limit[(int) DESCALE(tmp12 + tmp0,
CONST_BITS+PASS1_BITS+3+1)
& RANGE_MASK];
outptr[2] = range_limit[(int) DESCALE(tmp12 - tmp0,
CONST_BITS+PASS1_BITS+3+1)
& RANGE_MASK];

wsptr += DCTSIZE; /* advance pointer to next row */
}
}

/*
* Perform dequantization and inverse DCT on one block of coefficients,
* producing a reduced-size 2x2 output block.
*/

GLOBAL(void)
jpeg_idct_2x2 (j_decompress_ptr cinfo, jpeg_component_info * comptr,
JCOEFPTR coef_block,
JSAMPARRAY output_buf, JDIMENSION output_col)
{
    INT32 tmp0, tmp10, z1;
    JCOEFPTR inptr;
    ISLOW_MULT_TYPE * quantptr;
    int * wsptr;
    JSAMPROW outptr;
    JSAMPLE *range_limit = IDCT_range_limit(cinfo);
    int ctr;
    int workspace[DCTSIZE*2]; /* buffers data between passes */
    SHIFT_TEMPS

    /* Pass 1: process columns from input, store into work array. */
    inptr = coef_block;
    quantptr = (ISLOW_MULT_TYPE *) comptr->dct_table;
    wsptr = workspace;
    for (ctr = DCTSIZE; ctr > 0; inptr++, quantptr++, wsptr++, ctr--) {
        /* Don't bother to process columns 2,4,6 */
        if (ctr == DCTSIZE-2 || ctr == DCTSIZE-4 || ctr == DCTSIZE-6)
            continue;
        if (inptr[DCTSIZE*1] == 0 && inptr[DCTSIZE*3] == 0 &&
            inptr[DCTSIZE*5] == 0 && inptr[DCTSIZE*7] == 0) {
            /* AC terms all zero; we need not examine terms 2,4,6 for 2x2 output */
            int dcv = DEQUANTIZE(inptr[DCTSIZE*0], quantptr[DCTSIZE*0]) << PASS1_BITS;

            wsptr[DCTSIZE*0] = dcv;
            wsptr[DCTSIZE*1] = dcv;

            continue;
        }

        /* Even part */

        z1 = DEQUANTIZE(inptr[DCTSIZE*0], quantptr[DCTSIZE*0]);
        tmp10 = z1 << (CONST_BITS+2);

        /* Odd part */

        z1 = DEQUANTIZE(inptr[DCTSIZE*7], quantptr[DCTSIZE*7]);
        tmp0 = MULTIPLY(z1, - FIX_0_720959822); /* sqrt(2) * (c7-c5+c3-c1) */
        z1 = DEQUANTIZE(inptr[DCTSIZE*5], quantptr[DCTSIZE*5]);
        tmp0 += MULTIPLY(z1, FIX_0_850430095); /* sqrt(2) * (-c1+c3+c5+c7) */
        z1 = DEQUANTIZE(inptr[DCTSIZE*3], quantptr[DCTSIZE*3]);
        tmp0 += MULTIPLY(z1, - FIX_1_272758580); /* sqrt(2) * (-c1+c3-c5-c7) */
        z1 = DEQUANTIZE(inptr[DCTSIZE*1], quantptr[DCTSIZE*1]);
        tmp0 += MULTIPLY(z1, FIX_3_624509785); /* sqrt(2) * (c1+c3+c5+c7) */
    }
}

```

```

/* Final output stage */

wsptr[DCTSIZE*0] = (int) DESCALE(tmp10 + tmp0, CONST_BITS-PASS1_BITS+2);
wsptr[DCTSIZE*1] = (int) DESCALE(tmp10 - tmp0, CONST_BITS-PASS1_BITS+2);
}

/* Pass 2: process 2 rows from work array, store into output array. */

wsptr = workspace;
for (ctr = 0; ctr < 2; ctr++) {
    outptr = output_buf[ctr] + output_col;
    /* It's not clear whether a zero row test is worthwhile here ... */

#ifdef NO_ZERO_ROW_TEST
    if (wsptr[1] == 0 && wsptr[3] == 0 && wsptr[5] == 0 && wsptr[7] == 0) {
        /* AC terms all zero */
        JSAMPLE dcvval = range_limit[(int) DESCALE((INT32) wsptr[0], PASS1_BITS+3)
            & RANGE_MASK];

        outptr[0] = dcvval;
        outptr[1] = dcvval;

        wsptr += DCTSIZE;      /* advance pointer to next row */
        continue;
    }
#endif
}

/* Even part */

tmp10 = ((INT32) wsptr[0]) << (CONST_BITS+2);

/* Odd part */

tmp0 = MULTIPLY((INT32) wsptr[7], - FIX_0_720959822) /* sqrt(2) * (c7-c5+c3-c1) */
+ MULTIPLY((INT32) wsptr[5], FIX_0_850430095) /* sqrt(2) * (-c1+c3+c5+c7) */
+ MULTIPLY((INT32) wsptr[3], - FIX_1_272758580) /* sqrt(2) * (-c1+c3-c5-c7) */
+ MULTIPLY((INT32) wsptr[1], FIX_3_624509785); /* sqrt(2) * (c1+c3+c5+c7) */

/* Final output stage */

outptr[0] = range_limit[(int) DESCALE(tmp10 + tmp0,
    CONST_BITS+PASS1_BITS+3+2)
    & RANGE_MASK];
outptr[1] = range_limit[(int) DESCALE(tmp10 - tmp0,
    CONST_BITS+PASS1_BITS+3+2)
    & RANGE_MASK];

wsptr += DCTSIZE;      /* advance pointer to next row */

/*
 * Perform dequantization and inverse DCT on one block of coefficients,
 * producing a reduced-size 1x1 output block.
 */

GLOBAL(void)
jpeg_idct_1x1 (j_decompress_ptr cinfo, jpeg_component_info * compptr,
    JCOEFPTR coef_block,
    JSAMPARRAY output_buf, JDIMENSION output_col)
{
    int dcvval;
    ISLOW_MULT_TYPE * quantptr;
    JSAMPLE *range_limit = IDCT_range_limit(cinfo);
    SHIFT_TEMPS

    /* We hardly need an inverse DCT routine for this: just take the
     * average pixel value, which is one-eighth of the DC coefficient.
     */
    quantptr = (ISLOW_MULT_TYPE *) compptr->dct_table;
    dcvval = DEQUANTIZE(coef_block[0], quantptr[0]);
    dcvval = (int) DESCALE((INT32) dcvval, 3);

    output_buf[0][output_col] = range_limit[dcvval & RANGE_MASK];
}

#endif /* IDCT_SCALING_SUPPORTED */

```

```

/*
 * jmemmgr.c
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains the JPEG system-independent memory management
 * routines. This code is usable across a wide variety of machines; most
 * of the system dependencies have been isolated in a separate file.
 * The major functions provided here are:
 *   * pool-based allocation and freeing of memory;
 *   * policy decisions about how to divide available memory among the
 *     virtual arrays;
 *   * control logic for swapping virtual arrays between main memory and
 *     backing storage.
 * The separate system-dependent file provides the actual backing-storage
 * access code, and it contains the policy decision about how much total
 * main memory to use.
 * This file is system-dependent in the sense that some of its functions
 * are unnecessary in some systems. For example, if there is enough virtual
 * memory so that backing storage will never be used, much of the virtual
 * array control logic could be removed. (Of course, if you have that much
 * memory then you shouldn't care about a little bit of unused code...)
 */

#define JPEG_INTERNALS
#define AM_MEMORY_MANAGER /* we define jvirt_xarray_control structs */
#include "jinclude.h"
#include "jpeglib.h"
#include "jmemsys.h" /* import the system-dependent declarations */

#ifdef NO_GETENV
#ifndef HAVE_STDLIB_H /* <stdlib.h> should declare getenv() */
extern char * getenv JPP((const char * name));
#endif
#endif

/*
 * Some important notes:
 *   The allocation routines provided here must never return NULL.
 *   They should exit to error_exit if unsuccessful.
 *
 *   It's not a good idea to try to merge the sarray and barray routines,
 *   even though they are textually almost the same, because samples are
 *   usually stored as bytes while coefficients are shorts or ints. Thus,
 *   in machines where byte pointers have a different representation from
 *   word pointers, the resulting machine code could not be the same.
 */

/*
 * Many machines require storage alignment: longs must start on 4-byte
 * boundaries, doubles on 8-byte boundaries, etc. On such machines, malloc()
 * always returns pointers that are multiples of the worst-case alignment
 * requirement, and we had better do so too.
 * There isn't any really portable way to determine the worst-case alignment
 * requirement. This module assumes that the alignment requirement is
 * multiples of sizeof(ALIGN_TYPE).
 * By default, we define ALIGN_TYPE as double. This is necessary on some
 * workstations (where doubles really do need 8-byte alignment) and will work
 * fine on nearly everything. If your machine has lesser alignment needs,
 * you can save a few bytes by making ALIGN_TYPE smaller.
 * The only place I know of where this will NOT work is certain Macintosh
 * 680x0 compilers that define double as a 10-byte IEEE extended float.
 * Doing 10-byte alignment is counterproductive because longwords won't be
 * aligned well. Put "#define ALIGN_TYPE long" in jconfig.h if you have
 * such a compiler.
 */

#ifdef ALIGN_TYPE
/* so can override from jconfig.h */
#define ALIGN_TYPE double
#endif

/*
 * We allocate objects from "pools", where each pool is gotten with a single
 * request to jpeg_get_small() or jpeg_get_large(). There is no per-object
 * overhead within a pool, except for alignment padding. Each pool has a

```

```

* header with a link to the next pool of the same class.
* Small and large pool headers are identical except that the latter's
* link pointer must be FAR on 80x86 machines.
* Notice that the "real" header fields are union'ed with a dummy ALIGN_TYPE
* field. This forces the compiler to make SIZEOF(small_pool_hdr) a multiple
* of the alignment requirement of ALIGN_TYPE.
*/

```

```

typedef union small_pool_struct * small_pool_ptr;

```

```

typedef union small_pool_struct {
    struct {
        small_pool_ptr next;    /* next in list of pools */
        size_t bytes_used;      /* how many bytes already used within pool */
        size_t bytes_left;      /* bytes still available in this pool */
    } hdr;
    ALIGN_TYPE dummy;          /* included in union to ensure alignment */
} small_pool_hdr;

```

```

typedef union large_pool_struct * large_pool_ptr;

```

```

typedef union large_pool_struct {
    struct {
        large_pool_ptr next;    /* next in list of pools */
        size_t bytes_used;      /* how many bytes already used within pool */
        size_t bytes_left;      /* bytes still available in this pool */
    } hdr;
    ALIGN_TYPE dummy;          /* included in union to ensure alignment */
} large_pool_hdr;

```

```

/*
 * Here is the full definition of a memory manager object.
 */

```

```

typedef struct {
    struct jpeg_memory_mgr pub;    /* public fields */

    /* Each pool identifier (lifetime class) names a linked list of pools. */
    small_pool_ptr small_list[JPOOL_NUMPOOLS];
    large_pool_ptr large_list[JPOOL_NUMPOOLS];

    /* Since we only have one lifetime class of virtual arrays, only one
     * linked list is necessary (for each datatype). Note that the virtual
     * array control blocks being linked together are actually stored somewhere
     * in the small-pool list.
     */
    jvirt_sarray_ptr virt_sarray_list;
    jvirt_barray_ptr virt_barray_list;

    /* This counts total space obtained from jpeg_get_small/large */
    long total_space_allocated;

    /* alloc_sarray and alloc_barray set this value for use by virtual
     * array routines.
     */
    JDIMENSION last_rowsperchunk; /* from most recent alloc_sarray/barray */
} my_memory_mgr;

```

```

typedef my_memory_mgr * my_mem_ptr;

```

```

/*
 * The control blocks for virtual arrays.
 * Note that these blocks are allocated in the "small" pool area.
 * System-dependent info for the associated backing store (if any) is hidden
 * inside the backing_store_info struct.
 */

```

```

struct jvirt_sarray_control {
    JSAMPARRAY mem_buffer;    /* => the in-memory buffer */
    JDIMENSION rows_in_array; /* total virtual array height */
    JDIMENSION samplesperrow; /* width of array (and of memory buffer) */
    JDIMENSION maxaccess;     /* max rows accessed by access_virt_sarray */
    JDIMENSION rows_in_mem;    /* height of memory buffer */
    JDIMENSION rowsperchunk;   /* allocation chunk size in mem_buffer */
    JDIMENSION cur_start_row;  /* first logical row # in the buffer */
    JDIMENSION first_undef_row; /* row # of first uninitialized row */
    boolean pre_zero;          /* pre-zero mode requested? */
    boolean dirty;             /* do current buffer contents need written? */
}

```



```

boolean b_s_open; /* is backing-store data valid? */
jvirt_sarray_ptr next; /* link to next virtual sarray control block */
backing_store_info b_s_info; /* System-dependent control info */
};

struct jvirt_barray_control {
    JBLOCKARRAY mem_buffer; /* => the in-memory buffer */
    JDIMENSION rows_in_array; /* total virtual array height */
    JDIMENSION blocksperrow; /* width of array (and of memory buffer) */
    JDIMENSION maxaccess; /* max rows accessed by access_virt_barray */
    JDIMENSION rows_in_mem; /* height of memory buffer */
    JDIMENSION rowsperchunk; /* allocation chunk size in mem_buffer */
    JDIMENSION cur_start_row; /* first logical row # in the buffer */
    JDIMENSION first_undef_row; /* row # of first uninitialized row */
    boolean pre_zero; /* pre-zero mode requested? */
    boolean dirty; /* do current buffer contents need written? */
    boolean b_s_open; /* is backing-store data valid? */
    jvirt_barray_ptr next; /* link to next virtual barray control block */
    backing_store_info b_s_info; /* System-dependent control info */
};

#ifdef MEM_STATS /* optional extra stuff for statistics */

LOCAL(void)
print_mem_stats (j_common_ptr cinfo, int pool_id)
{
    my_mem_ptr mem = (my_mem_ptr) cinfo->mem;
    small_pool_ptr shdr_ptr;
    large_pool_ptr lhdr_ptr;

    /* Since this is only a debugging stub, we can cheat a little by using
     * fprintf directly rather than going through the trace message code.
     * This is helpful because message parm array can't handle longs.
     */
    fprintf(stderr, "Freeing pool %d, total space = %ld\n",
        pool_id, mem->total_space_allocated);

    for (lhdr_ptr = mem->large_list[pool_id]; lhdr_ptr != NULL;
        lhdr_ptr = lhdr_ptr->hdr.next) {
        fprintf(stderr, " Large chunk used %ld\n",
            (long) lhdr_ptr->hdr.bytes_used);
    }

    for (shdr_ptr = mem->small_list[pool_id]; shdr_ptr != NULL;
        shdr_ptr = shdr_ptr->hdr.next) {
        fprintf(stderr, " Small chunk used %ld free %ld\n",
            (long) shdr_ptr->hdr.bytes_used,
            (long) shdr_ptr->hdr.bytes_left);
    }
}

#endif /* MEM_STATS */

LOCAL(void)
out_of_memory (j_common_ptr cinfo, int which)
/* Report an out-of-memory error and stop execution */
/* If we compiled MEM_STATS support, report alloc requests before dying */
{
#ifdef MEM_STATS
    cinfo->err->trace_level = 2; /* force self_destruct to report stats */
#endif
    ERREXIT1(cinfo, JERR_OUT_OF_MEMORY, which);
}

/*
 * Allocation of "small" objects.
 *
 * For these, we use pooled storage. When a new pool must be created,
 * we try to get enough space for the current request plus a "slop" factor,
 * where the slop will be the amount of leftover space in the new pool.
 * The speed vs. space tradeoff is largely determined by the slop values.
 * A different slop value is provided for each pool class (lifetime),
 * and we also distinguish the first pool of a class from later ones.
 * NOTE: the values given work fairly well on both 16- and 32-bit-int
 * machines, but may be too small if longs are 64 bits or more.
 */

```

```

static const size_t first_pool_slop[JPOOL_NUMPOOLS] =
{
    1600,          /* first PERMANENT pool */
    16000         /* first IMAGE pool */
};

static const size_t extra_pool_slop[JPOOL_NUMPOOLS] =
{
    0,             /* additional PERMANENT pools */
    5000          /* additional IMAGE pools */
};

#define MIN_SLOP 50      /* greater than 0 to avoid futile looping */

METHODDEF(void *)
alloc_small (j_common_ptr cinfo, int pool_id, size_t sizeobject)
/* Allocate a "small" object */
{
    my_mem_ptr mem = (my_mem_ptr) cinfo->mem;
    small_pool_ptr hdr_ptr, prev_hdr_ptr;
    char * data_ptr;
    size_t odd_bytes, min_request, slop;

    /* Check for unsatisfiable request (do now to ensure no overflow below) */
    if (sizeobject > (size_t) (MAX_ALLOC_CHUNK-SIZEOF(small_pool_hdr)))
        out_of_memory(cinfo, 1); /* request exceeds malloc's ability */

    /* Round up the requested size to a multiple of SIZEOF(ALIGN_TYPE) */
    odd_bytes = sizeobject % SIZEOF(ALIGN_TYPE);
    if (odd_bytes > 0)
        sizeobject += SIZEOF(ALIGN_TYPE) - odd_bytes;

    /* See if space is available in any existing pool */
    if (pool_id < 0 || pool_id >= JPOOL_NUMPOOLS)
        ERREXIT1(cinfo, JERR_BAD_POOL_ID, pool_id); /* safety check */
    prev_hdr_ptr = NULL;
    hdr_ptr = mem->small_list[pool_id];
    while (hdr_ptr != NULL) {
        if (hdr_ptr->hdr.bytes_left >= sizeobject)
            break; /* found pool with enough space */
        prev_hdr_ptr = hdr_ptr;
        hdr_ptr = hdr_ptr->hdr.next;
    }

    /* Time to make a new pool? */
    if (hdr_ptr == NULL) {
        /* min_request is what we need now, slop is what will be leftover */
        min_request = sizeobject + SIZEOF(small_pool_hdr);
        if (prev_hdr_ptr == NULL) /* first pool in class? */
            slop = first_pool_slop[pool_id];
        else
            slop = extra_pool_slop[pool_id];
        /* Don't ask for more than MAX_ALLOC_CHUNK */
        if (slop > (size_t) (MAX_ALLOC_CHUNK-min_request))
            slop = (size_t) (MAX_ALLOC_CHUNK-min_request);
        /* Try to get space, if fail reduce slop and try again */
        for (;;) {
            hdr_ptr = (small_pool_ptr) jpeg_get_small(cinfo, min_request + slop);
            if (hdr_ptr != NULL)
                break;
            slop /= 2;
            if (slop < MIN_SLOP) /* give up when it gets real small */
                out_of_memory(cinfo, 2); /* jpeg_get_small failed */
        }
        mem->total_space_allocated += min_request + slop;
        /* Success, initialize the new pool header and add to end of list */
        hdr_ptr->hdr.next = NULL;
        hdr_ptr->hdr.bytes_used = 0;
        hdr_ptr->hdr.bytes_left = sizeobject + slop;
        if (prev_hdr_ptr == NULL) /* first pool in class? */
            mem->small_list[pool_id] = hdr_ptr;
        else
            prev_hdr_ptr->hdr.next = hdr_ptr;
    }

    /* OK, allocate the object from the current pool */
    data_ptr = (char *) (hdr_ptr + 1); /* point to first data byte in pool */
    data_ptr += hdr_ptr->hdr.bytes_used; /* point to place for object */
    hdr_ptr->hdr.bytes_used += sizeobject;

```

```

hdr_ptr->hdr.bytes_left -= sizeofobject;

return (void *) data_ptr;
}

/*
 * Allocation of "large" objects.
 *
 * The external semantics of these are the same as "small" objects,
 * except that FAR pointers are used on 80x86. However the pool
 * management heuristics are quite different. We assume that each
 * request is large enough that it may as well be passed directly to
 * jpeg_get_large; the pool management just links everything together
 * so that we can free it all on demand.
 * Note: the major use of "large" objects is in JSAMPARRAY and JBLOCKARRAY
 * structures. The routines that create these structures (see below)
 * deliberately bunch rows together to ensure a large request size.
 */

METHODDEF(void *)
alloc_large (j_common_ptr cinfo, int pool_id, size_t sizeofobject)
/* Allocate a "large" object */
{
    my_mem_ptr mem = (my_mem_ptr) cinfo->mem;
    large_pool_ptr hdr_ptr;
    size_t odd_bytes;

    /* Check for unsatisfiable request (do now to ensure no overflow below) */
    if (sizeofobject > (size_t) (MAX_ALLOC_CHUNK-SIZEOF(large_pool_hdr)))
        out_of_memory(cinfo, 3); /* request exceeds malloc's ability */

    /* Round up the requested size to a multiple of SIZEOF(ALIGN_TYPE) */
    odd_bytes = sizeofobject % SIZEOF(ALIGN_TYPE);
    if (odd_bytes > 0)
        sizeofobject += SIZEOF(ALIGN_TYPE) - odd_bytes;

    /* Always make a new pool */
    if (pool_id < 0 || pool_id >= JPOOL_NUMPOOLS)
        ERREXIT1(cinfo, JERR_BAD_POOL_ID, pool_id); /* safety check */

    hdr_ptr = (large_pool_ptr) jpeg_get_large(cinfo, sizeofobject +
        SIZEOF(large_pool_hdr));

    if (hdr_ptr == NULL)
        out_of_memory(cinfo, 4); /* jpeg_get_large failed */
    mem->total_space_allocated += sizeofobject + SIZEOF(large_pool_hdr);

    /* Success, initialize the new pool header and add to list */
    hdr_ptr->hdr.next = mem->large_list[pool_id];
    /* We maintain space counts in each pool header for statistical purposes,
     * even though they are not needed for allocation.
     */
    hdr_ptr->hdr.bytes_used = sizeofobject;
    hdr_ptr->hdr.bytes_left = 0;
    mem->large_list[pool_id] = hdr_ptr;

    return (void *) (hdr_ptr + 1); /* point to first data byte in pool */
}

/*
 * Creation of 2-D sample arrays.
 * The pointers are in near heap, the samples themselves in FAR heap.
 *
 * To minimize allocation overhead and to allow I/O of large contiguous
 * blocks, we allocate the sample rows in groups of as many rows as possible
 * without exceeding MAX_ALLOC_CHUNK total bytes per allocation request.
 * NB: the virtual array control routines, later in this file, know about
 * this chunking of rows. The rowsperchunk value is left in the mem manager
 * object so that it can be saved away if this sarray is the workspace for
 * a virtual array.
 */

METHODDEF(JSAMPARRAY)
alloc_sarray (j_common_ptr cinfo, int pool_id,
              JDIMENSION samplesperrow, JDIMENSION numrows)
/* Allocate a 2-D sample array */
{
    my_mem_ptr mem = (my_mem_ptr) cinfo->mem;
    JSAMPARRAY result;

```

```

JSAMPROW workspace;
JDIMENSION rowsperchunk, currow, i;
long ltemp;

/* Calculate max # of rows allowed in one allocation chunk */
ltemp = (MAX_ALLOC_CHUNK-SIZEOF(large_pool_hdr)) /
    ((long) samplesperrow * SIZEOF(JSAMPLE));
if (ltemp <= 0)
    ERREXIT(cinfo, JERR_WIDTH_OVERFLOW);
if (ltemp < (long) numRows)
    rowsperchunk = (JDIMENSION) ltemp;
else
    rowsperchunk = numRows;
mem->last_rowsperchunk = rowsperchunk;

/* Get space for row pointers (small object) */
result = (JSAMPARRAY) alloc_small(cinfo, pool_id,
    (size_t) (numRows * SIZEOF(JSAMPROW)));

/* Get the rows themselves (large objects) */
currow = 0;
while (currow < numRows) {
    rowsperchunk = MIN(rowsperchunk, numRows - currow);
    workspace = (JSAMPROW) alloc_large(cinfo, pool_id,
        (size_t) ((size_t) rowsperchunk * (size_t) samplesperrow
            * SIZEOF(JSAMPLE)));
    for (i = rowsperchunk; i > 0; i--) {
        result[currow++] = workspace;
        workspace += samplesperrow;
    }
}

return result;
}

/*
 * Creation of 2-D coefficient-block arrays.
 * This is essentially the same as the code for sample arrays, above.
 */

METHODDEF(JBLOCKARRAY)
alloc_barray (j_common_ptr cinfo, int pool_id,
    JDIMENSION blocksperrow, JDIMENSION numRows)
/* Allocate a 2-D coefficient-block array */
{
    my_mem_ptr mem = (my_mem_ptr) cinfo->mem;
    JOCKARRAY result;
    JOCKROW workspace;
    JDIMENSION rowsperchunk, currow, i;
    long ltemp;

/* Calculate max # of rows allowed in one allocation chunk */
ltemp = (MAX_ALLOC_CHUNK-SIZEOF(large_pool_hdr)) /
    ((long) blocksperrow * SIZEOF(JBLOCK));
if (ltemp <= 0)
    ERREXIT(cinfo, JERR_WIDTH_OVERFLOW);
if (ltemp < (long) numRows)
    rowsperchunk = (JDIMENSION) ltemp;
else
    rowsperchunk = numRows;
mem->last_rowsperchunk = rowsperchunk;

/* Get space for row pointers (small object) */
result = (JBLOCKARRAY) alloc_small(cinfo, pool_id,
    (size_t) (numRows * SIZEOF(JBLOCKROW)));

/* Get the rows themselves (large objects) */
currow = 0;
while (currow < numRows) {
    rowsperchunk = MIN(rowsperchunk, numRows - currow);
    workspace = (JBLOCKROW) alloc_large(cinfo, pool_id,
        (size_t) ((size_t) rowsperchunk * (size_t) blocksperrow
            * SIZEOF(JBLOCK)));
    for (i = rowsperchunk; i > 0; i--) {
        result[currow++] = workspace;
        workspace += blocksperrow;
    }
}
}

```

```

return result;
}

/*
 * About virtual array management:
 *
 * The above "normal" array routines are only used to allocate strip buffers
 * (as wide as the image, but just a few rows high). Full-image-sized buffers
 * are handled as "virtual" arrays. The array is still accessed a strip at a
 * time, but the memory manager must save the whole array for repeated
 * accesses. The intended implementation is that there is a strip buffer in
 * memory (as high as is possible given the desired memory limit), plus a
 * backing file that holds the rest of the array.
 *
 * The request_virt_array routines are told the total size of the image and
 * the maximum number of rows that will be accessed at once. The in-memory
 * buffer must be at least as large as the maxaccess value.
 *
 * The request routines create control blocks but not the in-memory buffers.
 * That is postponed until realize_virt_arrays is called. At that time the
 * total amount of space needed is known (approximately, anyway), so free
 * memory can be divided up fairly.
 *
 * The access_virt_array routines are responsible for making a specific strip
 * area accessible (after reading or writing the backing file, if necessary).
 * Note that the access routines are told whether the caller intends to modify
 * the accessed strip; during a read-only pass this saves having to rewrite
 * data to disk. The access routines are also responsible for pre-zeroing
 * any newly accessed rows, if pre-zeroing was requested.
 *
 * In current usage, the access requests are usually for nonoverlapping
 * strips; that is, successive access start_row numbers differ by exactly
 * num_rows = maxaccess. This means we can get good performance with simple
 * buffer dump/reload logic, by making the in-memory buffer be a multiple
 * of the access height; then there will never be accesses across bufferload
 * boundaries. The code will still work with overlapping access requests,
 * but it doesn't handle bufferload overlaps very efficiently.
 */

METHODDEF(jvirt_sarray_ptr)
request_virt_sarray (j_common_ptr cinfo, int pool_id, boolean pre_zero,
                    JDIMENSION samplesperrow, JDIMENSION numrows,
                    JDIMENSION maxaccess)
/* Request a virtual 2-D sample array */
{
    my_mem_ptr mem = (my_mem_ptr) cinfo->mem;
    jvirt_sarray_ptr result;

    /* Only IMAGE-lifetime virtual arrays are currently supported */
    if (pool_id != JPOOL_IMAGE)
        ERREXIT1(cinfo, JERR_BAD_POOL_ID, pool_id); /* safety check */

    /* get control block */
    result = (jvirt_sarray_ptr) alloc_small(cinfo, pool_id,
        sizeof(struct jvirt_sarray_control));

    result->mem_buffer = NULL; /* marks array not yet realized */
    result->rows_in_array = numrows;
    result->samplesperrow = samplesperrow;
    result->maxaccess = maxaccess;
    result->pre_zero = pre_zero;
    result->b_s_open = FALSE; /* no associated backing-store object */
    result->next = mem->virt_sarray_list; /* add to list of virtual arrays */
    mem->virt_sarray_list = result;

    return result;
}

METHODDEF(jvirt_barray_ptr)
request_virt_barray (j_common_ptr cinfo, int pool_id, boolean pre_zero,
                    JDIMENSION blocksperrow, JDIMENSION numrows,
                    JDIMENSION maxaccess)
/* Request a virtual 2-D coefficient-block array */
{
    my_mem_ptr mem = (my_mem_ptr) cinfo->mem;
    jvirt_barray_ptr result;

```

```

/* Only IMAGE-lifetime virtual arrays are currently supported */
if (pool_id != JPOOL_IMAGE)
    ERREXIT1(cinfo, JERR_BAD_POOL_ID, pool_id); /* safety check */

/* get control block */
result = (jvirt_barray_ptr) alloc_small(cinfo, pool_id,
    sizeof(struct jvirt_barray_control));

result->mem_buffer = NULL; /* marks array not yet realized */
result->rows_in_array = numrows;
result->blocksperrrow = blocksperrrow;
result->maxaccess = maxaccess;
result->pre_zero = pre_zero;
result->b_s_open = FALSE; /* no associated backing-store object */
result->next = mem->virt_barray_list; /* add to list of virtual arrays */
mem->virt_barray_list = result;

return result;
}

METHODDEF(void)
realize_virt_arrays (j_common_ptr cinfo)
/* Allocate the in-memory buffers for any unrealized virtual arrays */
{
    my_mem_ptr mem = (my_mem_ptr) cinfo->mem;
    long space_per_minheight, maximum_space, avail_mem;
    long minheights, max_minheights;
    jvirt_sarray_ptr sptr;
    jvirt_barray_ptr bptr;

    /* Compute the minimum space needed (maxaccess rows in each buffer)
     * and the maximum space needed (full image height in each buffer).
     * These may be of use to the system-dependent jpeg_mem_available routine.
     */
    space_per_minheight = 0;
    maximum_space = 0;
    for (sptr = mem->virt_sarray_list; sptr != NULL; sptr = sptr->next) {
        if (sptr->mem_buffer == NULL) { /* if not realized yet */
            space_per_minheight += (long) sptr->maxaccess *
                (long) sptr->samplesperrow * sizeof(JSAMPLE);
            maximum_space += (long) sptr->rows_in_array *
                (long) sptr->samplesperrow * sizeof(JSAMPLE);
        }
    }
    for (bptr = mem->virt_barray_list; bptr != NULL; bptr = bptr->next) {
        if (bptr->mem_buffer == NULL) { /* if not realized yet */
            space_per_minheight += (long) bptr->maxaccess *
                (long) bptr->blocksperrrow * sizeof(JBLOCK);
            maximum_space += (long) bptr->rows_in_array *
                (long) bptr->blocksperrrow * sizeof(JBLOCK);
        }
    }

    if (space_per_minheight <= 0)
        return; /* no unrealized arrays, no work */

    /* Determine amount of memory to actually use; this is system-dependent. */
    avail_mem = jpeg_mem_available(cinfo, space_per_minheight, maximum_space,
        mem->total_space_allocated);

    /* If the maximum space needed is available, make all the buffers full
     * height; otherwise parcel it out with the same number of minheights
     * in each buffer.
     */
    if (avail_mem >= maximum_space)
        max_minheights = 1000000000L;
    else {
        max_minheights = avail_mem / space_per_minheight;
        /* If there doesn't seem to be enough space, try to get the minimum
         * anyway. This allows a "stub" implementation of jpeg_mem_available().
         */
        if (max_minheights <= 0)
            max_minheights = 1;
    }

    /* Allocate the in-memory buffers and initialize backing store as needed. */
    for (sptr = mem->virt_sarray_list; sptr != NULL; sptr = sptr->next) {
        if (sptr->mem_buffer == NULL) { /* if not realized yet */

```

```

    minheights = ((long) sptr->rows_in_array - 1L) / sptr->maxaccess + 1L;
    if (minheights <= max_minheights) {
/* This buffer fits in memory */
    sptr->rows_in_mem = sptr->rows_in_array;
    } else {
/* It doesn't fit in memory, create backing store. */
    sptr->rows_in_mem = (JDIMENSION) (max_minheights * sptr->maxaccess);
    jpeg_open_backing_store(cinfo, & sptr->b_s_info,
        (long) sptr->rows_in_array *
        (long) sptr->samplesperrow *
        (long) SIZEOF(JSAMPLE));
    sptr->b_s_open = TRUE;
    }
    sptr->mem_buffer = alloc_sarray(cinfo, JPOOL_IMAGE,
        sptr->samplesperrow, sptr->rows_in_mem);
    sptr->rowsperchunk = mem->last_rowsperchunk;
    sptr->cur_start_row = 0;
    sptr->first_undef_row = 0;
    sptr->dirty = FALSE;
}

for (bptr = mem->virt_barray_list; bptr != NULL; bptr = bptr->next) {
    if (bptr->mem_buffer == NULL) { /* if not realized yet */
        minheights = ((long) bptr->rows_in_array - 1L) / bptr->maxaccess + 1L;
        if (minheights <= max_minheights) {
/* This buffer fits in memory */
            bptr->rows_in_mem = bptr->rows_in_array;
        } else {
/* It doesn't fit in memory, create backing store. */
            bptr->rows_in_mem = (JDIMENSION) (max_minheights * bptr->maxaccess);
            jpeg_open_backing_store(cinfo, & bptr->b_s_info,
                (long) bptr->rows_in_array *
                (long) bptr->blocksperrow *
                (long) SIZEOF(JBLOCK));
            bptr->b_s_open = TRUE;
        }
        bptr->mem_buffer = alloc_barray(cinfo, JPOOL_IMAGE,
            bptr->blocksperrow, bptr->rows_in_mem);
        bptr->rowsperchunk = mem->last_rowsperchunk;
        bptr->cur_start_row = 0;
        bptr->first_undef_row = 0;
        bptr->dirty = FALSE;
    }
}

LOCAL(void)
do_sarray_io (j_common_ptr cinfo, jvirt_sarray_ptr ptr, boolean writing)
/* Do backing store read or write of a virtual sample array */
{
    long bytesperrow, file_offset, byte_count, rows, thisrow, i;

    bytesperrow = (long) ptr->samplesperrow * SIZEOF(JSAMPLE);
    file_offset = ptr->cur_start_row * bytesperrow;
/* Loop to read or write each allocation chunk in mem_buffer */
    for (i = 0; i < (long) ptr->rows_in_mem; i += ptr->rowsperchunk) {
/* One chunk, but check for short chunk at end of buffer */
        rows = MIN((long) ptr->rowsperchunk, (long) ptr->rows_in_mem - i);
/* Transfer no more than is currently defined */
        thisrow = (long) ptr->cur_start_row + i;
        rows = MIN(rows, (long) ptr->first_undef_row - thisrow);
/* Transfer no more than fits in file */
        rows = MIN(rows, (long) ptr->rows_in_array - thisrow);
        if (rows <= 0) /* this chunk might be past end of file! */
            break;
        byte_count = rows * bytesperrow;
        if (writing)
            (*ptr->b_s_info.write_backing_store) (cinfo, & ptr->b_s_info,
                (void *) ptr->mem_buffer[i],
                file_offset, byte_count);
        else
            (*ptr->b_s_info.read_backing_store) (cinfo, & ptr->b_s_info,
                (void *) ptr->mem_buffer[i],
                file_offset, byte_count);
        file_offset += byte_count;
    }
}

```

```

LOCAL(void)
do_barray_io (j_common_ptr cinfo, jvirt_barray_ptr ptr, boolean writing)
/* Do backing store read or write of a virtual coefficient-block array */
{
    long bytesperrow, file_offset, byte_count, rows, thisrow, i;

    bytesperrow = (long) ptr->blocksperrow * SIZEOF(JBLOCK);
    file_offset = ptr->cur_start_row * bytesperrow;
    /* Loop to read or write each allocation chunk in mem_buffer */
    for (i = 0; i < (long) ptr->rows_in_mem; i += ptr->rowsperchunk) {
        /* One chunk, but check for short chunk at end of buffer */
        rows = MIN((long) ptr->rowsperchunk, (long) ptr->rows_in_mem - i);
        /* Transfer no more than is currently defined */
        thisrow = (long) ptr->cur_start_row + i;
        rows = MIN(rows, (long) ptr->first_undef_row - thisrow);
        /* Transfer no more than fits in file */
        rows = MIN(rows, (long) ptr->rows_in_array - thisrow);
        if (rows <= 0) /* this chunk might be past end of file! */
            break;
        byte_count = rows * bytesperrow;
        if (writing)
            (*ptr->b_s_info.write_backing_store) (cinfo, & ptr->b_s_info,
                                                  (void *) ptr->mem_buffer[i],
                                                  file_offset, byte_count);
        else
            (*ptr->b_s_info.read_backing_store) (cinfo, & ptr->b_s_info,
                                                  (void *) ptr->mem_buffer[i],
                                                  file_offset, byte_count);
        file_offset += byte_count;
    }
}

METHODDEF(JSAMPARRAY)
access_virt_sarray (j_common_ptr cinfo, jvirt_sarray_ptr ptr,
                    JDIMENSION start_row, JDIMENSION num_rows,
                    boolean writable)
/* Access the part of a virtual sample array starting at start_row */
/* and extending for num_rows rows. writable is true if */
/* caller intends to modify the accessed area. */
{
    JDIMENSION end_row = start_row + num_rows;
    JDIMENSION undef_row;

    /* debugging check */
    if (end_row > ptr->rows_in_array || num_rows > ptr->maxaccess ||
        ptr->mem_buffer == NULL)
        ERREXIT(cinfo, JERR_BAD_VIRTUAL_ACCESS);

    /* Make the desired part of the virtual array accessible */
    if (start_row < ptr->cur_start_row ||
        end_row > ptr->cur_start_row + ptr->rows_in_mem) {
        if (! ptr->b_s_open)
            ERREXIT(cinfo, JERR_VIRTUAL_BUG);
        /* Flush old buffer contents if necessary */
        if (ptr->dirty) {
            do_sarray_io(cinfo, ptr, TRUE);
            ptr->dirty = FALSE;
        }
        /* Decide what part of virtual array to access.
         * Algorithm: if target address > current window, assume forward scan,
         * load starting at target address. If target address < current window,
         * assume backward scan, load so that target area is top of window.
         * Note that when switching from forward write to forward read, will have
         * start_row = 0, so the limiting case applies and we load from 0 anyway.
         */
        if (start_row > ptr->cur_start_row) {
            ptr->cur_start_row = start_row;
        } else {
            /* use long arithmetic here to avoid overflow & unsigned problems */
            long ltemp;

            ltemp = (long) end_row - (long) ptr->rows_in_mem;
            if (ltemp < 0)
                ltemp = 0; /* don't fall off front end of file */
            ptr->cur_start_row = (JDIMENSION) ltemp;
        }
        /* Read in the selected part of the array.
         * During the initial write pass, we will do no actual read

```



```

    * because the selected part is all undefined.
    */
do_sarray_io(cinfo, ptr, FALSE);
}
/* Ensure the accessed part of the array is defined; prezero if needed.
 * To improve locality of access, we only prezero the part of the array
 * that the caller is about to access, not the entire in-memory array.
 */
if (ptr->first_undef_row < end_row) {
    if (ptr->first_undef_row < start_row) {
        if (writable) /* writer skipped over a section of array */
            ERREXIT(cinfo, JERR_BAD_VIRTUAL_ACCESS);
        undef_row = start_row; /* but reader is allowed to read ahead */
    } else {
        undef_row = ptr->first_undef_row;
    }
    if (writable)
        ptr->first_undef_row = end_row;
    if (ptr->pre_zero) {
        size_t bytesperrow = (size_t) ptr->samplesperrow * sizeof(JSAMPLE);
        undef_row -= ptr->cur_start_row; /* make indexes relative to buffer */
        end_row -= ptr->cur_start_row;
        while (undef_row < end_row) {
            jzero_far((void *) ptr->mem_buffer[undef_row], bytesperrow);
            undef_row++;
        }
    } else {
        if (!writable) /* reader looking at undefined data */
            ERREXIT(cinfo, JERR_BAD_VIRTUAL_ACCESS);
    }
}
/* Flag the buffer dirty if caller will write in it */
if (writable)
    ptr->dirty = TRUE;
/* Return address of proper part of the buffer */
return ptr->mem_buffer + (start_row - ptr->cur_start_row);
}

METHODDEF(JBLOCKARRAY)
access_virt_barray (j_common_ptr cinfo, jvirt_barray_ptr ptr,
                    JDIMENSION start_row, JDIMENSION num_rows,
                    boolean writable)
/* Access the part of a virtual block array starting at start_row */
/* and extending for num_rows rows. writable is true if */
/* caller intends to modify the accessed area. */
{
    JDIMENSION end_row = start_row + num_rows;
    JDIMENSION undef_row;

    /* debugging check */
    if (end_row > ptr->rows_in_array || num_rows > ptr->maxaccess ||
        ptr->mem_buffer == NULL)
        ERREXIT(cinfo, JERR_BAD_VIRTUAL_ACCESS);

    /* Make the desired part of the virtual array accessible */
    if (start_row < ptr->cur_start_row ||
        end_row > ptr->cur_start_row + ptr->rows_in_mem) {
        if (!ptr->b_s_open)
            ERREXIT(cinfo, JERR_VIRTUAL_BUG);
        /* Flush old buffer contents if necessary */
        if (ptr->dirty) {
            do_barray_io(cinfo, ptr, TRUE);
            ptr->dirty = FALSE;
        }
        /* Decide what part of virtual array to access.
         * Algorithm: if target address > current window, assume forward scan,
         * load starting at target address. If target address < current window,
         * assume backward scan, load so that target area is top of window.
         * Note that when switching from forward write to forward read, will have
         * start_row = 0, so the limiting case applies and we load from 0 anyway.
         */
        if (start_row > ptr->cur_start_row) {
            ptr->cur_start_row = start_row;
        } else {
            /* use long arithmetic here to avoid overflow & unsigned problems */
            long ltemp;

            ltemp = (long) end_row - (long) ptr->rows_in_mem;
            if (ltemp < 0)

```

```

ltemp = 0;      /* don't fall off front end of file */
ptr->cur_start_row = (JDIMENSION) ltemp;
}
/* Read in the selected part of the array.
 * During the initial write pass, we will do no actual read
 * because the selected part is all undefined.
 */
do_barray_io(cinfo, ptr, FALSE);
}
/* Ensure the accessed part of the array is defined; prezero if needed.
 * To improve locality of access, we only prezero the part of the array
 * that the caller is about to access, not the entire in-memory array.
 */
if (ptr->first_undef_row < end_row) {
  if (ptr->first_undef_row < start_row) {
    if (writable) /* writer skipped over a section of array */
      ERREXIT(cinfo, JERR_BAD_VIRTUAL_ACCESS);
    undef_row = start_row; /* but reader is allowed to read ahead */
  } else {
    undef_row = ptr->first_undef_row;
  }
  if (writable)
    ptr->first_undef_row = end_row;
  if (ptr->pre_zero) {
    size_t bytesperrow = (size_t) ptr->blocksprow * SIZEOF(JBLOCK);
    undef_row -= ptr->cur_start_row; /* make indexes relative to buffer */
    end_row -= ptr->cur_start_row;
    while (undef_row < end_row) {
      jzero_far((void *) ptr->mem_buffer[undef_row], bytesperrow);
      undef_row++;
    }
  } else {
    if (!writable) /* reader looking at undefined data */
      ERREXIT(cinfo, JERR_BAD_VIRTUAL_ACCESS);
  }
}
/* Flag the buffer dirty if caller will write in it */
if (writable)
  ptr->dirty = TRUE;
/* Return address of proper part of the buffer */
return ptr->mem_buffer + (start_row - ptr->cur_start_row);
}

/*
 * Release all objects belonging to a specified pool.
 */
METHODDEF(void)
free_pool (j_common_ptr cinfo, int pool_id)
{
  my_mem_ptr mem = (my_mem_ptr) cinfo->mem;
  small_pool_ptr shdr_ptr;
  large_pool_ptr lhdr_ptr;
  size_t space_freed;

  if (pool_id < 0 || pool_id >= JPOOL_NUMPOOLS)
    ERREXIT1(cinfo, JERR_BAD_POOL_ID, pool_id); /* safety check */

#ifdef MEM_STATS
  if (cinfo->err->trace_level > 1)
    print_mem_stats(cinfo, pool_id); /* print pool's memory usage statistics */
#endif

  /* If freeing IMAGE pool, close any virtual arrays first */
  if (pool_id == JPOOL_IMAGE) {
    jvirt_sarray_ptr sptr;
    jvirt_barray_ptr bptr;

    for (sptr = mem->virt_sarray_list; sptr != NULL; sptr = sptr->next) {
      if (sptr->b_s_open) { /* there may be no backing store */
        sptr->b_s_open = FALSE; /* prevent recursive close if error */
        (*sptr->b_s_info.close_backing_store) (cinfo, & sptr->b_s_info);
      }
    }
    mem->virt_sarray_list = NULL;
    for (bptr = mem->virt_barray_list; bptr != NULL; bptr = bptr->next) {
      if (bptr->b_s_open) { /* there may be no backing store */
        bptr->b_s_open = FALSE; /* prevent recursive close if error */
        (*bptr->b_s_info.close_backing_store) (cinfo, & bptr->b_s_info);
      }
    }
  }
}

```

```

    }
}
mem->virt_barray_list = NULL;
}

/* Release large objects */
lhdr_ptr = mem->large_list[pool_id];
mem->large_list[pool_id] = NULL;

while (lhdr_ptr != NULL) {
    large_pool_ptr next_lhdr_ptr = lhdr_ptr->hdr.next;
    space_freed = lhdr_ptr->hdr.bytes_used +
        lhdr_ptr->hdr.bytes_left +
        SIZEOF(large_pool_hdr);
    jpeg_free_large(cinfo, (void *) lhdr_ptr, space_freed);
    mem->total_space_allocated -= space_freed;
    lhdr_ptr = next_lhdr_ptr;
}

/* Release small objects */
shdr_ptr = mem->small_list[pool_id];
mem->small_list[pool_id] = NULL;

while (shdr_ptr != NULL) {
    small_pool_ptr next_shdr_ptr = shdr_ptr->hdr.next;
    space_freed = shdr_ptr->hdr.bytes_used +
        shdr_ptr->hdr.bytes_left +
        SIZEOF(small_pool_hdr);
    jpeg_free_small(cinfo, (void *) shdr_ptr, space_freed);
    mem->total_space_allocated -= space_freed;
    shdr_ptr = next_shdr_ptr;
}

}

Close up shop entirely.
* Note that this cannot be called unless cinfo->mem is non-NULL.
*/

METHODDEF(void)
self_destruct (j_common_ptr cinfo)
{
    int pool;

    /* Close all backing store, release all memory.
     * Releasing pools in reverse order might help avoid fragmentation
     * with some (brain-damaged) malloc libraries.
     */
    for (pool = JPOOL_NUMPOOLS-1; pool >= JPOOL_PERMANENT; pool--) {
        free_pool(cinfo, pool);
    }

    /* Release the memory manager control block too. */
    jpeg_free_small(cinfo, (void *) cinfo->mem, SIZEOF(my_memory_mgr));
    cinfo->mem = NULL;          /* ensures I will be called only once */

    jpeg_mem_term(cinfo);      /* system-dependent cleanup */
}

/*
 * Memory manager initialization.
 * When this is called, only the error manager pointer is valid in cinfo!
 */

GLOBAL(void)
jinit_memory_mgr (j_common_ptr cinfo)
{
    my_mem_ptr mem;
    long max_to_use;
    int pool;
    size_t test_mac;

    cinfo->mem = NULL;          /* for safety if init fails */

    /* Check for configuration errors.
     * SIZEOF(ALIGN_TYPE) should be a power of 2; otherwise, it probably
     * doesn't reflect any real hardware alignment requirement.
     * The test is a little tricky: for X>0, X and X-1 have no one-bits

```

```

    * in common if and only if X is a power of 2, ie has only one one-bit.
    * Some compilers may give an "unreachable code" warning here; ignore it.
    */
if ((sizeof(ALIGN_TYPE) & (sizeof(ALIGN_TYPE)-1)) != 0)
    ERREXIT(cinfo, JERR_BAD_ALIGN_TYPE);
/* MAX_ALLOC_CHUNK must be representable as type size_t, and must be
 * a multiple of sizeof(ALIGN_TYPE).
 * Again, an "unreachable code" warning may be ignored here.
 * But a "constant too large" warning means you need to fix MAX_ALLOC_CHUNK.
 */
test_mac = (size_t) MAX_ALLOC_CHUNK;
if ((long) test_mac != MAX_ALLOC_CHUNK ||
    (MAX_ALLOC_CHUNK % sizeof(ALIGN_TYPE)) != 0)
    ERREXIT(cinfo, JERR_BAD_ALLOC_CHUNK);

max_to_use = jpeg_mem_init(cinfo); /* system-dependent initialization */

/* Attempt to allocate memory manager's control block */
mem = (my_mem_ptr) jpeg_get_small(cinfo, sizeof(my_memory_mgr));

if (mem == NULL) {
    jpeg_mem_term(cinfo); /* system-dependent cleanup */
    ERREXIT1(cinfo, JERR_OUT_OF_MEMORY, 0);
}

/* OK, fill in the method pointers */
mem->pub.alloc_small = alloc_small;
mem->pub.alloc_large = alloc_large;
mem->pub.alloc_sarray = alloc_sarray;
mem->pub.alloc_barray = alloc_barray;
mem->pub.request_virt_sarray = request_virt_sarray;
mem->pub.request_virt_barray = request_virt_barray;
mem->pub.realize_virt_arrays = realize_virt_arrays;
mem->pub.access_virt_sarray = access_virt_sarray;
mem->pub.access_virt_barray = access_virt_barray;
mem->pub.free_pool = free_pool;
mem->pub.self_destruct = self_destruct;

/* Make MAX_ALLOC_CHUNK accessible to other modules */
mem->pub.max_alloc_chunk = MAX_ALLOC_CHUNK;

/* Initialize working state */
mem->pub.max_memory_to_use = max_to_use;

for (pool = JPOOL_NUMPOOLS-1; pool >= JPOOL_PERMANENT; pool--) {
    mem->small_list[pool] = NULL;
    mem->large_list[pool] = NULL;
}
mem->virt_sarray_list = NULL;
mem->virt_barray_list = NULL;

mem->total_space_allocated = sizeof(my_memory_mgr);

/* Declare ourselves open for business */
cinfo->mem = & mem->pub;

/* Check for an environment variable JPEGMEM; if found, override the
 * default max_memory setting from jpeg_mem_init. Note that the
 * surrounding application may again override this value.
 * If your system doesn't support getenv(), define NO_GETENV to disable
 * this feature.
 */
#ifdef NO_GETENV
{ char * memenv;

    if ((memenv = getenv("JPEGMEM")) != NULL) {
        char ch = 'x';

        if (sscanf(memenv, "%ld%c", &max_to_use, &ch) > 0) {
            if (ch == 'm' || ch == 'M')
                max_to_use *= 1000L;
            mem->pub.max_memory_to_use = max_to_use * 1000L;
        }
    }
}
#endif
}

```

```

/*
 * jmemnobs.c
 *
 * Copyright (C) 1992-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file provides a really simple implementation of the system-
 * dependent portion of the JPEG memory manager.  This implementation
 * assumes that no backing-store files are needed: all required space
 * can be obtained from malloc().
 * This is very portable in the sense that it'll compile on almost anything,
 * but you'd better have lots of main memory (or virtual memory) if you want
 * to process big images.
 * Note that the max_memory_to_use option is ignored by this implementation.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"
#include "jmemsys.h"          /* import the system-dependent declarations */

#ifdef HAVE_STDLIB_H          /* <stdlib.h> should declare malloc(),free() */
extern void * malloc JPP((size_t size));
extern void free JPP((void *ptr));
#endif

/*
 * Memory allocation and freeing are controlled by the regular library
 * routines malloc() and free().
 */

GLOBAL(void *)
jpeg_get_small (j_common_ptr cinfo, size_t sizeofobject)
{
  return (void *) malloc(sizeofobject);
}

GLOBAL(void)
jpeg_free_small (j_common_ptr cinfo, void * object, size_t sizeofobject)
{
  free(object);
}

/*
 * "Large" objects are treated the same as "small" ones.
 * NB: although we include FAR keywords in the routine declarations,
 * this file won't actually work in 80x86 small/medium model; at least,
 * you probably won't be able to process useful-size images in only 64KB.
 */

GLOBAL(void *)
jpeg_get_large (j_common_ptr cinfo, size_t sizeofobject)
{
  return (void *) malloc(sizeofobject);
}

GLOBAL(void)
jpeg_free_large (j_common_ptr cinfo, void * object, size_t sizeofobject)
{
  free(object);
}

/*
 * This routine computes the total memory space available for allocation.
 * Here we always say, "we got all you want bud!"
 */

GLOBAL(long)
jpeg_mem_available (j_common_ptr cinfo, long min_bytes_needed,
                   long max_bytes_needed, long already_allocated)
{
  return max_bytes_needed;
}

/*

```

```

* Backing store (temporary file) management.
* Since jpeg_mem_available always promised the moon,
* this should never be called and we can just error out.
*/

```

```

GLOBAL(void)
jpeg_open_backing_store (j_common_ptr cinfo, backing_store_ptr info,
                        long total_bytes_needed)
{
    ERREXIT(cinfo, JERR_NO_BACKING_STORE);
}

```

```

/*
* These routines take care of any system-dependent initialization and
* cleanup required. Here, there isn't any.
*/

```

```

GLOBAL(long)
jpeg_mem_init (j_common_ptr cinfo)
{
    return 0;          /* just set max_memory_to_use to 0 */
}

```

```

GLOBAL(void)
jpeg_mem_term (j_common_ptr cinfo)
{
    /* no work */
}

```

```

/*
 * jquant1.c
 *
 * Copyright (C) 1991-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains 1-pass color quantization (color mapping) routines.
 * These routines provide mapping to a fixed color map using equally spaced
 * color values. Optional Floyd-Steinberg or ordered dithering is available.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

#ifdef QUANT_1PASS_SUPPORTED

/*
 * The main purpose of 1-pass quantization is to provide a fast, if not very
 * high quality, colormapped output capability. A 2-pass quantizer usually
 * gives better visual quality; however, for quantized grayscale output this
 * quantizer is perfectly adequate. Dithering is highly recommended with this
 * quantizer, though you can turn it off if you really want to.
 *
 * In 1-pass quantization the colormap must be chosen in advance of seeing the
 * image. We use a map consisting of all combinations of Ncolors[i] color
 * values for the i'th component. The Ncolors[] values are chosen so that
 * their product, the total number of colors, is no more than that requested.
 * (In most cases, the product will be somewhat less.)
 *
 * Since the colormap is orthogonal, the representative value for each color
 * component can be determined without considering the other components;
 * then these indexes can be combined into a colormap index by a standard
 * N-dimensional-array-subscript calculation. Most of the arithmetic involved
 * can be precalculated and stored in the lookup table colorindex[].
 * colorindex[i][j] maps pixel value j in component i to the nearest
 * representative value (grid plane) for that component; this index is
 * multiplied by the array stride for component i, so that the
 * index of the colormap entry closest to a given pixel value is just
 * sum( colorindex[component-number][pixel-component-value] )
 * Aside from being fast, this scheme allows for variable spacing between
 * representative values with no additional lookup cost.
 *
 * If gamma correction has been applied in color conversion, it might be wise
 * to adjust the color grid spacing so that the representative colors are
 * equidistant in linear space. At this writing, gamma correction is not
 * implemented by jdcolor, so nothing is done here.
 */

/* Declarations for ordered dithering.
 *
 * We use a standard 16x16 ordered dither array. The basic concept of ordered
 * dithering is described in many references, for instance Dale Schumacher's
 * chapter II.2 of Graphics Gems II (James Arvo, ed. Academic Press, 1991).
 * In place of Schumacher's comparisons against a "threshold" value, we add a
 * "dither" value to the input pixel and then round the result to the nearest
 * output value. The dither value is equivalent to (0.5 - threshold) times
 * the distance between output values. For ordered dithering, we assume that
 * the output colors are equally spaced; if not, results will probably be
 * worse, since the dither may be too much or too little at a given point.
 *
 * The normal calculation would be to form pixel value + dither, range-limit
 * this to 0..MAXJSAMPLE, and then index into the colorindex table as usual.
 * We can skip the separate range-limiting step by extending the colorindex
 * table in both directions.
 */

#define ODITHER_SIZE 16 /* dimension of dither matrix */
/* NB: if ODITHER_SIZE is not a power of 2, ODITHER_MASK uses will break */
#define ODITHER_CELLS (ODITHER_SIZE*ODITHER_SIZE) /* # cells in matrix */
#define ODITHER_MASK (ODITHER_SIZE-1) /* mask for wrapping around counters */

typedef int ODITHER_MATRIX[ODITHER_SIZE][ODITHER_SIZE];
typedef int (*ODITHER_MATRIX_PTR)[ODITHER_SIZE];

static const UINT8 base_dither_matrix[ODITHER_SIZE][ODITHER_SIZE] = {
/* Bayer's order-4 dither array. Generated by the code given in

```

```

* Stephen Hawley's article "Ordered Dithering" in Graphics Gems I.
* The values in this array must range from 0 to ODITHER_CELLS-1.
*/
{ 0,192, 48,240, 12,204, 60,252, 3,195, 51,243, 15,207, 63,255 },
{ 128, 64,176,112,140, 76,188,124,131, 67,179,115,143, 79,191,127 },
{ 32,224, 16,208, 44,236, 28,220, 35,227, 19,211, 47,239, 31,223 },
{ 160, 96,144, 80,172,108,156, 92,163, 99,147, 83,175,111,159, 95 },
{ 8,200, 56,248, 4,196, 52,244, 11,203, 59,251, 7,199, 55,247 },
{ 136, 72,184,120,132, 68,180,116,139, 75,187,123,135, 71,183,119 },
{ 40,232, 24,216, 36,228, 20,212, 43,235, 27,219, 39,231, 23,215 },
{ 168,104,152, 88,164,100,148, 84,171,107,155, 91,167,103,151, 87 },
{ 2,194, 50,242, 14,206, 62,254, 1,193, 49,241, 13,205, 61,253 },
{ 130, 66,178,114,142, 78,190,126,129, 65,177,113,141, 77,189,125 },
{ 34,226, 18,210, 46,238, 30,222, 33,225, 17,209, 45,237, 29,221 },
{ 162, 98,146, 82,174,110,158, 94,161, 97,145, 81,173,109,157, 93 },
{ 10,202, 58,250, 6,198, 54,246, 9,201, 57,249, 5,197, 53,245 },
{ 138, 74,186,122,134, 70,182,118,137, 73,185,121,133, 69,181,117 },
{ 42,234, 26,218, 38,230, 22,214, 41,233, 25,217, 37,229, 21,213 },
{ 170,106,154, 90,166,102,150, 86,169,105,153, 89,165,101,149, 85 }
};

/* Declarations for Floyd-Steinberg dithering.
*
* Errors are accumulated into the array fserrors[], at a resolution of
* 1/16th of a pixel count. The error at a given pixel is propagated
* to its not-yet-processed neighbors using the standard F-S fractions,
* ... (here) 7/16
*           3/16 5/16 1/16
* We work left-to-right on even rows, right-to-left on odd rows.
*
* We can get away with a single array (holding one row's worth of errors)
* by using it to store the current row's errors at pixel columns not yet
* processed, but the next row's errors at columns already processed. We
* need only a few extra variables to hold the errors immediately around the
* current column. (If we are lucky, those variables are in registers, but
* even if not, they're probably cheaper to access than array elements are.)
*
* The fserrors[] array is indexed [component#][position].
* We provide (#columns + 2) entries per component; the extra entry at each
* end saves us from special-casing the first and last pixels.
*
* Note: on a wide image, we might not have enough room in a PC's near data
* segment to hold the error array; so it is allocated with alloc_large.
*/
#ifdef BITS_IN_JSAMPLE == 8
typedef INT16 FSERROR; /* 16 bits should be enough */
typedef int LOCFSEERROR; /* use 'int' for calculation temps */
#else
typedef INT32 FSERROR; /* may need more than 16 bits */
typedef INT32 LOCFSEERROR; /* be sure calculation temps are big enough */
#endif

typedef FSERROR *FSERRPTR; /* pointer to error array (in FAR storage!) */

/* Private subobject */

#define MAX_Q_COMPS 4 /* max components I can handle */

typedef struct {
    struct jpeg_color_quantizer pub; /* public fields */

    /* Initially allocated colormap is saved here */
    JSAMPARRAY sv_colormap; /* The color map as a 2-D pixel array */
    int sv_actual; /* number of entries in use */

    JSAMPARRAY colorindex; /* Precomputed mapping for speed */
    /* colorindex[i][j] = index of color closest to pixel value j in component i,
     * premultiplied as described above. Since colormap indexes must fit into
     * JSAMPLEs, the entries of this array will too.
     */
    boolean is_padded; /* is the colorindex padded for odither? */

    int Ncolors[MAX_Q_COMPS]; /* # of values allocated to each component */

    /* Variables for ordered dithering */
    int row_index; /* cur row's vertical index in dither matrix */
    ODITHER_MATRIX_PTR odither[MAX_Q_COMPS]; /* one dither array per component */

```



```

/* Variables for Floyd-Steinberg dithering */
FSERRPTR ferrors[MAX_Q_COMPS]; /* accumulated errors */
boolean on_odd_row; /* flag to remember which row we are on */
} my_cquantizer;

typedef my_cquantizer * my_cquantize_ptr;

/*
 * Policy-making subroutines for create_colormap and create_colorindex.
 * These routines determine the colormap to be used. The rest of the module
 * only assumes that the colormap is orthogonal.
 *
 * * select_ncolors decides how to divvy up the available colors
 *   among the components.
 * * output_value defines the set of representative values for a component.
 * * largest_input_value defines the mapping from input values to
 *   representative values for a component.
 * Note that the latter two routines may impose different policies for
 * different components, though this is not currently done.
 */

LOCAL(int)
select_ncolors (j_decompress_ptr cinfo, int Ncolors[])
/* Determine allocation of desired colors to components, */
/* and fill in Ncolors[] array to indicate choice. */
/* Return value is total number of colors (product of Ncolors[] values). */
{
    int nc = cinfo->out_color_components; /* number of color components */
    int max_colors = cinfo->desired_number_of_colors;
    int total_colors, iroot, i, j;
    boolean changed;
    long temp;
    static const int RGB_order[3] = { RGB_GREEN, RGB_RED, RGB_BLUE };

    /* We can allocate at least the nc'th root of max_colors per component. */
    /* Compute floor(nc'th root of max_colors). */
    iroot = 1;
    do {
        iroot++;
        temp = iroot; /* set temp = iroot ** nc */
        for (i = 1; i < nc; i++)
            temp *= iroot;
    } while (temp <= (long) max_colors); /* repeat till iroot exceeds root */
    iroot--; /* now iroot = floor(root) */

    /* Must have at least 2 color values per component */
    if (iroot < 2)
        ERREXIT1(cinfo, JERR_QUANT_FEW_COLORS, (int) temp);

    /* Initialize to iroot color values for each component */
    total_colors = 1;
    for (i = 0; i < nc; i++) {
        Ncolors[i] = iroot;
        total_colors *= iroot;
    }

    /* We may be able to increment the count for one or more components without
     * exceeding max_colors, though we know not all can be incremented.
     * Sometimes, the first component can be incremented more than once!
     * (Example: for 16 colors, we start at 2*2*2, go to 3*2*2, then 4*2*2.)
     * In RGB colorspace, try to increment G first, then R, then B.
     */
    do {
        changed = FALSE;
        for (i = 0; i < nc; i++) {
            j = (cinfo->out_color_space == JCS_RGB ? RGB_order[i] : i);
            /* calculate new total_colors if Ncolors[j] is incremented */
            temp = total_colors / Ncolors[j];
            temp *= Ncolors[j] + 1; /* done in long arith to avoid oflo */
            if (temp > (long) max_colors)
                break; /* won't fit, done with this pass */
            Ncolors[j]++; /* OK, apply the increment */
            total_colors = (int) temp;
            changed = TRUE;
        }
    } while (changed);

    return total_colors;
}

```

```

}

LOCAL(int)
output_value (j_decompress_ptr cinfo, int ci, int j, int maxj)
/* Return j'th output value, where j will range from 0 to maxj */
/* The output values must fall in 0..MAXJSAMPLE in increasing order */
{
    /* We always provide values 0 and MAXJSAMPLE for each component;
     * any additional values are equally spaced between these limits.
     * (Forcing the upper and lower values to the limits ensures that
     * dithering can't produce a color outside the selected gamut.)
     */
    return (int) (((INT32) j * MAXJSAMPLE + maxj/2) / maxj);
}

LOCAL(int)
largest_input_value (j_decompress_ptr cinfo, int ci, int j, int maxj)
/* Return largest input value that should map to j'th output value */
/* Must have largest(j=0) >= 0, and largest(j=maxj) >= MAXJSAMPLE */
{
    /* Breakpoints are halfway between values returned by output_value */
    return (int) (((INT32) (2*j + 1) * MAXJSAMPLE + maxj) / (2*maxj));
}

/*
 * Create the colormap.
 */
LOCAL(void)
create_colormap (j_decompress_ptr cinfo)
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    JSAMPARRAY colormap; /* Created colormap */
    int total_colors; /* Number of distinct output colors */
    int i,j,k, nci, blksize, blkdist, ptr, val;

    /* Select number of colors for each component */
    total_colors = select_ncolors(cinfo, cquantize->Ncolors);

    /* Report selected color counts */
    if (cinfo->out_color_components == 3)
        TRACE4(cinfo, 1, JTRC_QUANT_3_NCOLORS,
            total_colors, cquantize->Ncolors[0],
            cquantize->Ncolors[1], cquantize->Ncolors[2]);
    else
        TRACE1(cinfo, 1, JTRC_QUANT_NCOLORS, total_colors);

    /* Allocate and fill in the colormap. */
    /* The colors are ordered in the map in standard row-major order, */
    /* i.e. rightmost (highest-indexed) color changes most rapidly. */

    colormap = (*cinfo->mem->alloc_sarray)
        ((j_common_ptr) cinfo, JPOOL_IMAGE,
        (JDIMENSION) total_colors, (JDIMENSION) cinfo->out_color_components);

    /* blksize is number of adjacent repeated entries for a component */
    /* blkdist is distance between groups of identical entries for a component */
    blkdist = total_colors;

    for (i = 0; i < cinfo->out_color_components; i++) {
        /* fill in colormap entries for i'th color component */
        nci = cquantize->Ncolors[i]; /* # of distinct values for this color */
        blksize = blkdist / nci;
        for (j = 0; j < nci; j++) {
            /* Compute j'th output value (out of nci) for component */
            val = output_value(cinfo, i, j, nci-1);
            /* Fill in all colormap entries that have this value of this component */
            for (ptr = j * blksize; ptr < total_colors; ptr += blkdist) {
                /* fill in blksize entries beginning at ptr */
                for (k = 0; k < blksize; k++)
                    colormap[i][ptr+k] = (JSAMPLE) val;
            }
            blkdist = blksize; /* blksize of this color is blkdist of next */
        }
    }

    /* Save the colormap in private storage,

```

```

    * where it will survive color quantization mode changes.
    */
    cquantize->sv_colormap = colormap;
    cquantize->sv_actual = total_colors;
}

/*
 * Create the color index table.
 */

LOCAL(void)
create_colorindex (j_decompress_ptr cinfo)
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    JSAMPROW indexptr;
    int i,j,k, nci, blksize, val, pad;

    /* For ordered dither, we pad the color index tables by MAXJSAMPLE in
     * each direction (input index values can be -MAXJSAMPLE .. 2*MAXJSAMPLE).
     * This is not necessary in the other dithering modes. However, we
     * flag whether it was done in case user changes dithering mode.
     */
    if (cinfo->dither_mode == JDITHER_ORDERED) {
        pad = MAXJSAMPLE*2;
        cquantize->is_padded = TRUE;
    } else {
        pad = 0;
        cquantize->is_padded = FALSE;
    }

    cquantize->colorindex = (*cinfo->mem->alloc_sarray)
        ((j_common_ptr) cinfo, JPOOL_IMAGE,
         (JDIMENSION) (MAXJSAMPLE+1 + pad),
         (JDIMENSION) cinfo->out_color_components);

    /* blksize is number of adjacent repeated entries for a component */
    blksize = cquantize->sv_actual;

    for (i = 0; i < cinfo->out_color_components; i++) {
        /* fill in colorindex entries for i'th color component */
        nci = cquantize->Ncolors[i]; /* # of distinct values for this color */
        blksize = blksize / nci;

        /* adjust colorindex pointers to provide padding at negative indexes. */
        if (pad)
            cquantize->colorindex[i] += MAXJSAMPLE;

        /* in loop, val = index of current output value, */
        /* and k = largest j that maps to current val */
        indexptr = cquantize->colorindex[i];
        val = 0;
        k = largest_input_value(cinfo, i, 0, nci-1);
        for (j = 0; j <= MAXJSAMPLE; j++) {
            while (j > k) /* advance val if past boundary */
                k = largest_input_value(cinfo, i, ++val, nci-1);
            /* premultiply so that no multiplication needed in main processing */
            indexptr[j] = (JSAMPLE) (val * blksize);
        }
        /* Pad at both ends if necessary */
        if (pad)
            for (j = 1; j <= MAXJSAMPLE; j++) {
                indexptr[-j] = indexptr[0];
                indexptr[MAXJSAMPLE+j] = indexptr[MAXJSAMPLE];
            }
    }
}

/*
 * Create an ordered-dither array for a component having ncolors
 * distinct output values.
 */

LOCAL(ODITHER_MATRIX_PTR)
make_odither_array (j_decompress_ptr cinfo, int ncolors)
{
    ODITHER_MATRIX_PTR odither;
    int j,k;
    INT32 num, den;

```

```

odither = (ODITHER_MATRIX_PTR)
(*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
    sizeof(ODITHER_MATRIX));
/* The inter-value distance for this color is MAXJSAMPLE/(ncolors-1).
 * Hence the dither value for the matrix cell with fill order f
 * (f=0..N-1) should be (N-1-2*f)/(2*N) * MAXJSAMPLE/(ncolors-1).
 * On 16-bit-int machine, be careful to avoid overflow.
 */
den = 2 * ODITHER_CELLS * ((INT32) (ncolors - 1));
for (j = 0; j < ODITHER_SIZE; j++) {
    for (k = 0; k < ODITHER_SIZE; k++) {
        num = ((INT32) (ODITHER_CELLS-1 - 2*((int)base_dither_matrix[j][k])))
            * MAXJSAMPLE;
        /* Ensure round towards zero despite C's lack of consistency
         * about rounding negative values in integer division...
         */
        odither[j][k] = (int) (num<0 ? -((-num)/den) : num/den);
    }
}
return odither;
}

```

```

/*
 * Create the ordered-dither tables.
 * Components having the same number of representative colors may
 * share a dither table.
 */

```

```

LOCAL(void)
create_odither_tables (j_decompress_ptr cinfo)
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    ODITHER_MATRIX_PTR odither;
    int i, j, nci;

    for (i = 0; i < cinfo->out_color_components; i++) {
        nci = cquantize->Ncolors[i]; /* # of distinct values for this color */
        odither = NULL; /* search for matching prior component */
        for (j = 0; j < i; j++) {
            if (nci == cquantize->Ncolors[j]) {
                odither = cquantize->odither[j];
                break;
            }
        }
        if (odither == NULL) /* need a new table? */
            odither = make_odither_array(cinfo, nci);
        cquantize->odither[i] = odither;
    }
}

```

```

/*
 * Map some rows of pixels to the output colormapped representation.
 */

```

```

METHODDEF(void)
color_quantize (j_decompress_ptr cinfo, JSAMPARRAY input_buf,
    JSAMPARRAY output_buf, int num_rows)
/* General case, no dithering */
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    JSAMPARRAY colorindex = cquantize->colorindex;
    register int pixcode, ci;
    register JSAMPROW ptrin, ptrout;
    int row;
    JDIMENSION col;
    JDIMENSION width = cinfo->output_width;
    register int nc = cinfo->out_color_components;

    for (row = 0; row < num_rows; row++) {
        ptrin = input_buf[row];
        ptrout = output_buf[row];
        for (col = width; col > 0; col--) {
            pixcode = 0;
            for (ci = 0; ci < nc; ci++) {
                pixcode += GETJSAMPLE(colorindex[ci][GETJSAMPLE(*ptrin++)]);
            }
            *ptrout++ = (JSAMPLE) pixcode;
        }
    }
}

```

```

    }
}
}

METHODDEF(void)
color_quantize3 (j_decompress_ptr cinfo, JSAMPARRAY input_buf,
                 JSAMPARRAY output_buf, int num_rows)
/* Fast path for out_color_components==3, no dithering */
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    register int pixcode;
    register JSAMPROW ptrin, ptrout;
    JSAMPROW colorindex0 = cquantize->colorindex[0];
    JSAMPROW colorindex1 = cquantize->colorindex[1];
    JSAMPROW colorindex2 = cquantize->colorindex[2];
    int row;
    JDIMENSION col;
    JDIMENSION width = cinfo->output_width;

    for (row = 0; row < num_rows; row++) {
        ptrin = input_buf[row];
        ptrout = output_buf[row];
        for (col = width; col > 0; col--) {
            pixcode = GETJSAMPLE(colorindex0[GETJSAMPLE(*ptrin++)]);
            pixcode += GETJSAMPLE(colorindex1[GETJSAMPLE(*ptrin++)]);
            pixcode += GETJSAMPLE(colorindex2[GETJSAMPLE(*ptrin++)]);
            *ptrout++ = (JSAMPLE) pixcode;
        }
    }
}

```

```

METHODDEF(void)
color_quantize_ord_dither (j_decompress_ptr cinfo, JSAMPARRAY input_buf,
                          JSAMPARRAY output_buf, int num_rows)
/* General case, with ordered dithering */
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    register JSAMPROW input_ptr;
    register JSAMPROW output_ptr;
    JSAMPROW colorindex_ci;
    int * dither; /* points to active row of dither matrix */
    int row_index, col_index; /* current indexes into dither matrix */
    int nc = cinfo->out_color_components;
    int ci;
    int row;
    JDIMENSION col;
    JDIMENSION width = cinfo->output_width;

    for (row = 0; row < num_rows; row++) {
        /* Initialize output values to 0 so can process components separately */
        jzero_far((void *) output_buf[row],
                 (size_t) (width * SIZEOF(JSAMPLE)));
        row_index = cquantize->row_index;
        for (ci = 0; ci < nc; ci++) {
            input_ptr = input_buf[row] + ci;
            output_ptr = output_buf[row];
            colorindex_ci = cquantize->colorindex[ci];
            dither = cquantize->odither[ci][row_index];
            col_index = 0;

            for (col = width; col > 0; col--) {
                /* Form pixel value + dither, range-limit to 0..MAXJSAMPLE,
                 * select output value, accumulate into output code for this pixel.
                 * Range-limiting need not be done explicitly, as we have extended
                 * the colorindex table to produce the right answers for out-of-range
                 * inputs. The maximum dither is +- MAXJSAMPLE; this sets the
                 * required amount of padding.
                 */
                *output_ptr += colorindex_ci[GETJSAMPLE(*input_ptr)+dither[col_index]];
                input_ptr += nc;
                output_ptr++;
                col_index = (col_index + 1) & ODITHER_MASK;
            }
            /* Advance row index for next row */
            row_index = (row_index + 1) & ODITHER_MASK;
            cquantize->row_index = row_index;
        }
    }
}

```

```

}

METHODDEF(void)
quantize3_ord_dither (j_decompress_ptr cinfo, JSAMPARRAY input_buf,
                      JSAMPARRAY output_buf, int num_rows)
/* Fast path for out_color_components==3, with ordered dithering */
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    register int pixcode;
    register JSAMPROW input_ptr;
    register JSAMPROW output_ptr;
    JSAMPROW colorindex0 = cquantize->colorindex[0];
    JSAMPROW colorindex1 = cquantize->colorindex[1];
    JSAMPROW colorindex2 = cquantize->colorindex[2];
    int * dither0; /* points to active row of dither matrix */
    int * dither1;
    int * dither2;
    int row_index, col_index; /* current indexes into dither matrix */
    int row;
    JDIMENSION col;
    JDIMENSION width = cinfo->output_width;

    for (row = 0; row < num_rows; row++) {
        row_index = cquantize->row_index;
        input_ptr = input_buf[row];
        output_ptr = output_buf[row];
        dither0 = cquantize->odither[0][row_index];
        dither1 = cquantize->odither[1][row_index];
        dither2 = cquantize->odither[2][row_index];
        col_index = 0;

        for (col = width; col > 0; col--) {
            pixcode = GETJSAMPLE(colorindex0[GETJSAMPLE(*input_ptr++) +
                                         dither0[col_index]]);
            pixcode += GETJSAMPLE(colorindex1[GETJSAMPLE(*input_ptr++) +
                                         dither1[col_index]]);
            pixcode += GETJSAMPLE(colorindex2[GETJSAMPLE(*input_ptr++) +
                                         dither2[col_index]]);
            *output_ptr++ = (JSAMPLE) pixcode;
            col_index = (col_index + 1) & ODITHER_MASK;
        }
        row_index = (row_index + 1) & ODITHER_MASK;
        cquantize->row_index = row_index;
    }
}

```

```

METHODDEF(void)
quantize_fs_dither (j_decompress_ptr cinfo, JSAMPARRAY input_buf,
                    JSAMPARRAY output_buf, int num_rows)
/* General case, with Floyd-Steinberg dithering */
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    register LOCFSEERROR cur; /* current error or pixel value */
    LOCFSEERROR belowerr; /* error for pixel below cur */
    LOCFSEERROR bpreverr; /* error for below/prev col */
    LOCFSEERROR bnexterr; /* error for below/next col */
    LOCFSEERROR delta;
    register FSERRPTR errorptr; /* => fseerrors[] at column before current */
    register JSAMPROW input_ptr;
    register JSAMPROW output_ptr;
    JSAMPROW colorindex_ci;
    JSAMPROW colormap_ci;
    int pixcode;
    int nc = cinfo->out_color_components;
    int dir; /* 1 for left-to-right, -1 for right-to-left */
    int dirnc; /* dir * nc */
    int ci;
    int row;
    JDIMENSION col;
    JDIMENSION width = cinfo->output_width;
    JSAMPLE *range_limit = cinfo->sample_range_limit;
    SHIFT_TEMPS

    for (row = 0; row < num_rows; row++) {
        /* Initialize output values to 0 so can process components separately */
        jzero_far((void *) output_buf[row],
                  (size_t) (width * SIZEOF(JSAMPLE)));
        for (ci = 0; ci < nc; ci++) {

```

```

    input_ptr = input_buf[row] + ci;
    output_ptr = output_buf[row];
    if (cquantize->on_odd_row) {
/* work right to left in this row */
input_ptr += (width-1) * nc; /* so point to rightmost pixel */
output_ptr += width-1;
dir = -1;
dirnc = -nc;
errorptr = cquantize->ferrors[ci] + (width+1); /* => entry after last column */
    } else {
/* work left to right in this row */
dir = 1;
dirnc = nc;
errorptr = cquantize->ferrors[ci]; /* => entry before first column */
    }
    colorindex_ci = cquantize->colorindex[ci];
    colormap_ci = cquantize->sv_colormap[ci];
    /* Preset error values: no error propagated to first pixel from left */
    cur = 0;
    /* and no error propagated to row below yet */
    belowerr = bpreverr = 0;

    for (col = width; col > 0; col--) {
/* cur holds the error propagated from the previous pixel on the
* current line. Add the error propagated from the previous line
* to form the complete error correction term for this pixel, and
* round the error term (which is expressed * 16) to an integer.
* RIGHT_SHIFT rounds towards minus infinity, so adding 8 is correct
* for either sign of the error value.
* Note: errorptr points to *previous* column's array entry.
*/
cur = RIGHT_SHIFT(cur + errorptr[dir] + 8, 4);
/* Form pixel value + error, and range-limit to 0..MAXJSAMPLE.
* The maximum error is +- MAXJSAMPLE; this sets the required size
* of the range_limit array.
*/
cur += GETJSAMPLE(*input_ptr);
cur = GETJSAMPLE(range_limit[cur]);
/* Select output value, accumulate into output code for this pixel */
pixcode = GETJSAMPLE(colorindex_ci[cur]);
*output_ptr += (JSAMPLE) pixcode;
/* Compute actual representation error at this pixel */
/* Note: we can do this even though we don't have the final */
/* pixel code, because the colormap is orthogonal. */
cur -= GETJSAMPLE(colormap_ci[pixcode]);
/* Compute error fractions to be propagated to adjacent pixels.
* Add these into the running sums, and simultaneously shift the
* next-line error sums left by 1 column.
*/
bnexterr = cur;
delta = cur * 2;
cur += delta; /* form error * 3 */
errorptr[0] = (FSERROR) (bpreverr + cur);
cur += delta; /* form error * 5 */
bpreverr = belowerr + cur;
belowerr = bnexterr;
cur += delta; /* form error * 7 */
/* At this point cur contains the 7/16 error value to be propagated
* to the next pixel on the current line, and all the errors for the
* next line have been shifted over. We are therefore ready to move on.
*/
input_ptr += dirnc; /* advance input ptr to next column */
output_ptr += dir; /* advance output ptr to next column */
errorptr += dir; /* advance errorptr to current column */
    }
    /* Post-loop cleanup: we must unload the final error value into the
    * final ferrors[] entry. Note we need not unload belowerr because
    * it is for the dummy column before or after the actual array.
    */
    errorptr[0] = (FSERROR) bpreverr; /* unload prev err into array */
}
cquantize->on_odd_row = (cquantize->on_odd_row ? FALSE : TRUE);
}
}

/*
* Allocate workspace for Floyd-Steinberg errors.
*/

```

```

LOCAL(void)
alloc_fs_workspace (j_decompress_ptr cinfo)
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    size_t arraysize;
    int i;

    arraysize = (size_t) ((cinfo->output_width + 2) * SIZEOF(FSERROR));
    for (i = 0; i < cinfo->out_color_components; i++) {
        cquantize->fserrors[i] = (FSERRPTR)
            (*cinfo->mem->alloc_large)((j_common_ptr) cinfo, JPOOL_IMAGE, arraysize);
    }
}

/*
 * Initialize for one-pass color quantization.
 */

METHODDEF(void)
start_pass_1_quant (j_decompress_ptr cinfo, boolean is_pre_scan)
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    size_t arraysize;
    int i;

    /* Install my colormap. */
    cinfo->colormap = cquantize->sv_colormap;
    cinfo->actual_number_of_colors = cquantize->sv_actual;

    /* Initialize for desired dithering mode. */
    switch (cinfo->dither_mode) {
        case JDITHER_NONE:
            if (cinfo->out_color_components == 3)
                cquantize->pub.color_quantize = color_quantize3;
            else
                cquantize->pub.color_quantize = color_quantize;
            break;
        case JDITHER_ORDERED:
            if (cinfo->out_color_components == 3)
                cquantize->pub.color_quantize = quantize3_ord_dither;
            else
                cquantize->pub.color_quantize = quantize_ord_dither;
            cquantize->row_index = 0; /* initialize state for ordered dither */
            /* If user changed to ordered dither from another mode,
             * we must recreate the color index table with padding.
             * This will cost extra space, but probably isn't very likely.
             */
            if (! cquantize->is_padded)
                create_colorindex(cinfo);
            /* Create ordered-dither tables if we didn't already. */
            if (cquantize->odither[0] == NULL)
                create_odither_tables(cinfo);
            break;
        case JDITHER_FS:
            cquantize->pub.color_quantize = quantize_fs_dither;
            cquantize->on_odd_row = FALSE; /* initialize state for F-S dither */
            /* Allocate Floyd-Steinberg workspace if didn't already. */
            if (cquantize->fserrors[0] == NULL)
                alloc_fs_workspace(cinfo);
            /* Initialize the propagated errors to zero. */
            arraysize = (size_t) ((cinfo->output_width + 2) * SIZEOF(FSERROR));
            for (i = 0; i < cinfo->out_color_components; i++)
                jzero_far((void *) cquantize->fserrors[i], arraysize);
            break;
        default:
            ERREXIT(cinfo, JERR_NOT_COMPILED);
            break;
    }
}

/*
 * Finish up at the end of the pass.
 */

METHODDEF(void)
finish_pass_1_quant (j_decompress_ptr cinfo)
{
    /* no work in 1-pass case */
}

```



```

}

/*
 * Switch to a new external colormap between output passes.
 * Shouldn't get to this module!
 */

METHODDEF(void)
new_color_map_1_quant (j_decompress_ptr cinfo)
{
    ERREXIT(cinfo, JERR_MODE_CHANGE);
}

/*
 * Module initialization routine for 1-pass color quantization.
 */

GLOBAL(void)
jinit_1pass_quantizer (j_decompress_ptr cinfo)
{
    my_cquantize_ptr cquantize;

    cquantize = (my_cquantize_ptr)
        (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
            SIZEOF(my_cquantizer));
    cinfo->cquantize = (struct jpeg_color_quantizer *) cquantize;
    cquantize->pub.start_pass = start_pass_1_quant;
    cquantize->pub.finish_pass = finish_pass_1_quant;
    cquantize->pub.new_color_map = new_color_map_1_quant;
    cquantize->fserrors[0] = NULL; /* Flag FS workspace not allocated */
    cquantize->odither[0] = NULL; /* Also flag odither arrays not allocated */

    /* Make sure my internal arrays won't overflow */
    if (cinfo->out_color_components > MAX_Q_COMPS)
        ERREXIT1(cinfo, JERR_QUANT_COMPONENTS, MAX_Q_COMPS);
    /* Make sure colormap indexes can be represented by JSAMPLEs */
    if (cinfo->desired_number_of_colors > (MAXJSAMPLE+1))
        ERREXIT1(cinfo, JERR_QUANT_MANY_COLORS, MAXJSAMPLE+1);

    /* Create the colormap and color index table. */
    create_colormap(cinfo);
    create_colorindex(cinfo);

    /* Allocate Floyd-Steinberg workspace now if requested.
     * We do this now since it is FAR storage and may affect the memory
     * manager's space calculations.  If the user changes to FS dither
     * mode in a later pass, we will allocate the space then, and will
     * possibly overrun the max_memory_to_use setting.
     */
    if (cinfo->dither_mode == JDITHER_FS)
        alloc_fs_workspace(cinfo);
}

#endif /* QUANT_1PASS_SUPPORTED */

```

```

/*
 * jquant2.c
 *
 * Copyright (C) 1991-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains 2-pass color quantization (color mapping) routines.
 * These routines provide selection of a custom color map for an image,
 * followed by mapping of the image to that color map, with optional
 * Floyd-Steinberg dithering.
 * It is also possible to use just the second pass to map to an arbitrary
 * externally-given color map.
 *
 * Note: ordered dithering is not supported, since there isn't any fast
 * way to compute intercolor distances; it's unclear that ordered dither's
 * fundamental assumptions even hold with an irregularly spaced color map.
 */

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

#ifdef QUANT_2PASS_SUPPORTED

/*
 * This module implements the well-known Heckbert paradigm for color
 * quantization. Most of the ideas used here can be traced back to
 * Heckbert's seminal paper
 *   Heckbert, Paul. "Color Image Quantization for Frame Buffer Display",
 *   Proc. SIGGRAPH '82, Computer Graphics v.16 #3 (July 1982), pp 297-304.
 *
 * In the first pass over the image, we accumulate a histogram showing the
 * usage count of each possible color. To keep the histogram to a reasonable
 * size, we reduce the precision of the input; typical practice is to retain
 * 5 or 6 bits per color, so that 8 or 4 different input values are counted
 * in the same histogram cell.
 *
 * Next, the color-selection step begins with a box representing the whole
 * color space, and repeatedly splits the "largest" remaining box until we
 * have as many boxes as desired colors. Then the mean color in each
 * remaining box becomes one of the possible output colors.
 *
 * The second pass over the image maps each input pixel to the closest output
 * color (optionally after applying a Floyd-Steinberg dithering correction).
 * This mapping is logically trivial, but making it go fast enough requires
 * considerable care.
 *
 * Heckbert-style quantizers vary a good deal in their policies for choosing
 * the "largest" box and deciding where to cut it. The particular policies
 * used here have proved out well in experimental comparisons, but better ones
 * may yet be found.
 *
 * In earlier versions of the IJG code, this module quantized in YCbCr color
 * space, processing the raw upsampled data without a color conversion step.
 * This allowed the color conversion math to be done only once per colormap
 * entry, not once per pixel. However, that optimization precluded other
 * useful optimizations (such as merging color conversion with upsampling)
 * and it also interfered with desired capabilities such as quantizing to an
 * externally-supplied colormap. We have therefore abandoned that approach.
 * The present code works in the post-conversion color space, typically RGB.
 *
 * To improve the visual quality of the results, we actually work in scaled
 * RGB space, giving G distances more weight than R, and R in turn more than
 * B. To do everything in integer math, we must use integer scale factors.
 * The 2/3/1 scale factors used here correspond loosely to the relative
 * weights of the colors in the NTSC grayscale equation.
 * If you want to use this code to quantize a non-RGB color space, you'll
 * probably need to change these scale factors.
 */

#define R_SCALE 2      /* scale R distances by this much */
#define G_SCALE 3      /* scale G distances by this much */
#define B_SCALE 1      /* and B by this much */

/* Relabel R/G/B as components 0/1/2, respecting the RGB ordering defined
 * in jmorecfg.h. As the code stands, it will do the right thing for R,G,B
 * and B,G,R orders. If you define some other weird order in jmorecfg.h,
 * you'll get compile errors until you extend this logic. In that case

```

```

* you'll probably want to tweak the histogram sizes too.
*/

#if RGB_RED == 0
#define C0_SCALE R_SCALE
#endif
#if RGB_BLUE == 0
#define C0_SCALE B_SCALE
#endif
#if RGB_GREEN == 1
#define C1_SCALE G_SCALE
#endif
#if RGB_RED == 2
#define C2_SCALE R_SCALE
#endif
#if RGB_BLUE == 2
#define C2_SCALE B_SCALE
#endif

/*
 * First we have the histogram data structure and routines for creating it.
 *
 * The number of bits of precision can be adjusted by changing these symbols.
 * We recommend keeping 6 bits for G and 5 each for R and B.
 * If you have plenty of memory and cycles, 6 bits all around gives marginally
 * better results; if you are short of memory, 5 bits all around will save
 * some space but degrade the results.
 * To maintain a fully accurate histogram, we'd need to allocate a "long"
 * (preferably unsigned long) for each cell. In practice this is overkill;
 * we can get by with 16 bits per cell. Few of the cell counts will overflow,
 * and clamping those that do overflow to the maximum value will give close-
 * enough results. This reduces the recommended histogram size from 256Kb
 * to 128Kb, which is a useful savings on PC-class machines.
 * (In the second pass the histogram space is re-used for pixel mapping data;
 * in that capacity, each cell must be able to store zero to the number of
 * desired colors. 16 bits/cell is plenty for that too.)
 * Since the JPEG code is intended to run in small memory model on 80x86
 * machines, we can't just allocate the histogram in one chunk. Instead
 * of a true 3-D array, we use a row of pointers to 2-D arrays. Each
 * pointer corresponds to a C0 value (typically 2^5 = 32 pointers) and
 * each 2-D array has 2^6*2^5 = 2048 or 2^6*2^6 = 4096 entries. Note that
 * on 80x86 machines, the pointer row is in near memory but the actual
 * arrays are in far memory (same arrangement as we use for image arrays).
 */
#define MAXNUMCOLORS (MAXJSAMPLE+1) /* maximum size of colormap */

/* These will do the right thing for either R,G,B or B,G,R color order,
 * but you may not like the results for other color orders.
 */
#define HIST_C0_BITS 5 /* bits of precision in R/B histogram */
#define HIST_C1_BITS 6 /* bits of precision in G histogram */
#define HIST_C2_BITS 5 /* bits of precision in B/R histogram */

/* Number of elements along histogram axes. */
#define HIST_C0_ELEMS (1<<HIST_C0_BITS)
#define HIST_C1_ELEMS (1<<HIST_C1_BITS)
#define HIST_C2_ELEMS (1<<HIST_C2_BITS)

/* These are the amounts to shift an input value to get a histogram index. */
#define C0_SHIFT (BITS_IN_JSAMPLE-HIST_C0_BITS)
#define C1_SHIFT (BITS_IN_JSAMPLE-HIST_C1_BITS)
#define C2_SHIFT (BITS_IN_JSAMPLE-HIST_C2_BITS)

typedef UINT16 histcell; /* histogram cell; prefer an unsigned type */
typedef histcell * histptr; /* for pointers to histogram cells */

typedef histcell hist1d[HIST_C2_ELEMS]; /* typedefs for the array */
typedef hist1d * hist2d; /* type for the 2nd-level pointers */
typedef hist2d * hist3d; /* type for top-level pointer */

/* Declarations for Floyd-Steinberg dithering.
 *
 * Errors are accumulated into the array ferrors[], at a resolution of
 * 1/16th of a pixel count. The error at a given pixel is propagated
 * to its not-yet-processed neighbors using the standard F-S fractions,

```

```

*      ... (here) 7/16
*      3/16      5/16      1/16
* We work left-to-right on even rows, right-to-left on odd rows.
*
* We can get away with a single array (holding one row's worth of errors)
* by using it to store the current row's errors at pixel columns not yet
* processed, but the next row's errors at columns already processed. We
* need only a few extra variables to hold the errors immediately around the
* current column. (If we are lucky, those variables are in registers, but
* even if not, they're probably cheaper to access than array elements are.)
*
* The ferrors[] array has (#columns + 2) entries; the extra entry at
* each end saves us from special-casing the first and last pixels.
* Each entry is three values long, one value for each color component.
*
* Note: on a wide image, we might not have enough room in a PC's near data
* segment to hold the error array; so it is allocated with alloc_large.
*/

#if BITS_IN_JSAMPLE == 8
typedef INT16 FSERROR;      /* 16 bits should be enough */
typedef int LOCFSError;     /* use 'int' for calculation temps */
#else
typedef INT32 FSERROR;      /* may need more than 16 bits */
typedef INT32 LOCFSError;   /* be sure calculation temps are big enough */
#endif

typedef FSERROR *FSERRPTR; /* pointer to error array (in FAR storage!) */

/* Private subobject */
typedef struct {
    struct jpeg_color_quantizer pub; /* public fields */

    /* Space for the eventually created colormap is stashed here */
    JSAMPARRAY sv_colormap; /* colormap allocated at init time */
    int desired; /* desired # of colors = size of colormap */

    /* Variables for accumulating image statistics */
    hist3d histogram; /* pointer to the histogram */

    boolean needs_zeroed; /* TRUE if next pass must zero histogram */

    /* Variables for Floyd-Steinberg dithering */
    FSERRPTR ferrors; /* accumulated errors */
    boolean on_odd_row; /* flag to remember which row we are on */
    int * error_limiter; /* table for clamping the applied error */
} my_quantizer;

typedef my_quantizer * my_quantize_ptr;

/*
 * Prescan some rows of pixels.
 * In this module the prescan simply updates the histogram, which has been
 * initialized to zeros by start_pass.
 * An output_buf parameter is required by the method signature, but no data
 * is actually output (in fact the buffer controller is probably passing a
 * NULL pointer).
 */

METHODDEF(void)
prescan_quantize(j_decompress_ptr cinfo, JSAMPARRAY input_buf,
                 JSAMPARRAY output_buf, int num_rows)
{
    my_quantize_ptr cquantize = (my_quantize_ptr) cinfo->cquantize;
    register JSAMPROW ptr;
    register histptr histp;
    register hist3d histogram = cquantize->histogram;
    int row;
    JDIMENSION col;
    JDIMENSION width = cinfo->output_width;

    for (row = 0; row < num_rows; row++) {
        ptr = input_buf[row];
        for (col = width; col > 0; col--) {
            /* get pixel value and index into the histogram */
            histp = & histogram[GETJSAMPLE(ptr[0]) >> C0_SHIFT]
                [GETJSAMPLE(ptr[1]) >> C1_SHIFT]

```

```

        [GETJSAMPLE(ptr[2]) >> C2_SHIFT];
/* increment, check for overflow and undo increment if so. */
if (++(*histp) <= 0)
    (*histp)--;
    ptr += 3;
}
}

/*
 * Next we have the really interesting routines: selection of a colormap
 * given the completed histogram.
 * These routines work with a list of "boxes", each representing a rectangular
 * subset of the input color space (to histogram precision).
 */

typedef struct {
/* The bounds of the box (inclusive); expressed as histogram indexes */
    int c0min, c0max;
    int c1min, c1max;
    int c2min, c2max;
/* The volume (actually 2-norm) of the box */
    INT32 volume;
/* The number of nonzero histogram cells within this box */
    long colorcount;
} box;

typedef box * boxptr;

LOCAL(boxptr)
find_biggest_color_pop (boxptr boxlist, int numboxes)
/* Find the splittable box with the largest color population */
/* Returns NULL if no splittable boxes remain */
{
    register boxptr boxp;
    register int i;
    register long maxc = 0;
    boxptr which = NULL;

    for (i = 0, boxp = boxlist; i < numboxes; i++, boxp++) {
        if (boxp->colorcount > maxc && boxp->volume > 0) {
            which = boxp;
            maxc = boxp->colorcount;
        }
    }
    return which;
}

LOCAL(boxptr)
find_biggest_volume (boxptr boxlist, int numboxes)
/* Find the splittable box with the largest (scaled) volume */
/* Returns NULL if no splittable boxes remain */
{
    register boxptr boxp;
    register int i;
    register INT32 maxv = 0;
    boxptr which = NULL;

    for (i = 0, boxp = boxlist; i < numboxes; i++, boxp++) {
        if (boxp->volume > maxv) {
            which = boxp;
            maxv = boxp->volume;
        }
    }
    return which;
}

LOCAL(void)
update_box (j_decompress_ptr cinfo, boxptr boxp)
/* Shrink the min/max bounds of a box to enclose only nonzero elements, */
/* and recompute its volume and population */
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    hist3d histogram = cquantize->histogram;
    histptr histp;
    int c0, c1, c2;

```

```

int c0min,c0max,c1min,c1max,c2min,c2max;
INT32 dist0,dist1,dist2;
long ccount;

c0min = boxp->c0min; c0max = boxp->c0max;
c1min = boxp->c1min; c1max = boxp->c1max;
c2min = boxp->c2min; c2max = boxp->c2max;

if (c0max > c0min)
  for (c0 = c0min; c0 <= c0max; c0++)
    for (c1 = c1min; c1 <= c1max; c1++) {
      histp = & histogram[c0][c1][c2min];
      for (c2 = c2min; c2 <= c2max; c2++)
        if (*histp++ != 0) {
          boxp->c0min = c0min = c0;
          goto have_c0min;
        }
    }
have_c0min:
if (c0max > c0min)
  for (c0 = c0max; c0 >= c0min; c0--)
    for (c1 = c1min; c1 <= c1max; c1++) {
      histp = & histogram[c0][c1][c2min];
      for (c2 = c2min; c2 <= c2max; c2++)
        if (*histp++ != 0) {
          boxp->c0max = c0max = c0;
          goto have_c0max;
        }
    }
have_c0max:
if (c1max > c1min)
  for (c1 = c1min; c1 <= c1max; c1++)
    for (c0 = c0min; c0 <= c0max; c0++) {
      histp = & histogram[c0][c1][c2min];
      for (c2 = c2min; c2 <= c2max; c2++)
        if (*histp++ != 0) {
          boxp->c1min = c1min = c1;
          goto have_c1min;
        }
    }
have_c1min:
if (c1max > c1min)
  for (c1 = c1max; c1 >= c1min; c1--)
    for (c0 = c0min; c0 <= c0max; c0++) {
      histp = & histogram[c0][c1][c2min];
      for (c2 = c2min; c2 <= c2max; c2++)
        if (*histp++ != 0) {
          boxp->c1max = c1max = c1;
          goto have_c1max;
        }
    }
have_c1max:
if (c2max > c2min)
  for (c2 = c2min; c2 <= c2max; c2++)
    for (c0 = c0min; c0 <= c0max; c0++) {
      histp = & histogram[c0][c1min][c2];
      for (c1 = c1min; c1 <= c1max; c1++, histp += HIST_C2_ELEMS)
        if (*histp != 0) {
          boxp->c2min = c2min = c2;
          goto have_c2min;
        }
    }
have_c2min:
if (c2max > c2min)
  for (c2 = c2max; c2 >= c2min; c2--)
    for (c0 = c0min; c0 <= c0max; c0++) {
      histp = & histogram[c0][c1min][c2];
      for (c1 = c1min; c1 <= c1max; c1++, histp += HIST_C2_ELEMS)
        if (*histp != 0) {
          boxp->c2max = c2max = c2;
          goto have_c2max;
        }
    }
have_c2max:

/* Update box volume
 * We use 2-norm rather than real volume here; this biases the method
 * against making long narrow boxes, and it has the side benefit that
 * a box is splittable iff norm > 0.
 * Since the differences are expressed in histogram-cell units,

```

```

    * we have to shift back to JSAMPLE units to get consistent distances;
    * after which, we scale according to the selected distance scale factors.
    */
    dist0 = ((c0max - c0min) << C0_SHIFT) * C0_SCALE;
    dist1 = ((c1max - c1min) << C1_SHIFT) * C1_SCALE;
    dist2 = ((c2max - c2min) << C2_SHIFT) * C2_SCALE;
    boxp->volume = dist0*dist0 + dist1*dist1 + dist2*dist2;

    /* Now scan remaining volume of box and compute population */
    ccount = 0;
    for (c0 = c0min; c0 <= c0max; c0++)
        for (c1 = c1min; c1 <= c1max; c1++) {
            histp = & histogram[c0][c1][c2min];
            for (c2 = c2min; c2 <= c2max; c2++, histp++)
                if (*histp != 0) {
                    ccount++;
                }
        }
    boxp->colorcount = ccount;
}

LOCAL(int)
median_cut (j_decompress_ptr cinfo, boxptr boxlist, int numboxes,
            int desired_colors)
/* Repeatedly select and split the largest box until we have enough boxes */
{
    int n, lb;
    int c0, c1, c2, cmax;
    register boxptr b1, b2;

    while (numboxes < desired_colors) {
        /* Select box to split.
         * Current algorithm: by population for first half, then by volume.
         */
        if (numboxes*2 <= desired_colors) {
            b1 = find_biggest_color_pop(boxlist, numboxes);
        } else {
            b1 = find_biggest_volume(boxlist, numboxes);
        }
        if (b1 == NULL) /* no splittable boxes left! */
            break;
        b2 = &boxlist[numboxes]; /* where new box will go */
        /* Copy the color bounds to the new box. */
        b2->c0max = b1->c0max; b2->c1max = b1->c1max; b2->c2max = b1->c2max;
        b2->c0min = b1->c0min; b2->c1min = b1->c1min; b2->c2min = b1->c2min;
        /* Choose which axis to split the box on.
         * Current algorithm: longest scaled axis.
         * See notes in update_box about scaling distances.
         */
        c0 = ((b1->c0max - b1->c0min) << C0_SHIFT) * C0_SCALE;
        c1 = ((b1->c1max - b1->c1min) << C1_SHIFT) * C1_SCALE;
        c2 = ((b1->c2max - b1->c2min) << C2_SHIFT) * C2_SCALE;
        /* We want to break any ties in favor of green, then red, blue last.
         * This code does the right thing for R,G,B or B,G,R color orders only.
         */
        #if RGB_RED == 0
            cmax = c1; n = 1;
            if (c0 > cmax) { cmax = c0; n = 0; }
            if (c2 > cmax) { n = 2; }
        #else
            cmax = c1; n = 1;
            if (c2 > cmax) { cmax = c2; n = 2; }
            if (c0 > cmax) { n = 0; }
        #endif
        /* Choose split point along selected axis, and update box bounds.
         * Current algorithm: split at halfway point.
         * (Since the box has been shrunk to minimum volume,
         * any split will produce two nonempty subboxes.)
         * Note that lb value is max for lower box, so must be < old max.
         */
        switch (n) {
            case 0:
                lb = (b1->c0max + b1->c0min) / 2;
                b1->c0max = lb;
                b2->c0min = lb+1;
                break;
            case 1:
                lb = (b1->c1max + b1->c1min) / 2;
                b1->c1max = lb;

```

```

        b2->c1min = lb+1;
        break;
    case 2:
        lb = (b1->c2max + b1->c2min) / 2;
        b1->c2max = lb;
        b2->c2min = lb+1;
        break;
    }
    /* Update stats for boxes */
    update_box(cinfo, b1);
    update_box(cinfo, b2);
    numboxes++;
}
return numboxes;
}

```

```

LOCAL(void)
compute_color (j_decompress_ptr cinfo, boxptr boxp, int icolor)
/* Compute representative color for a box, put it in colormap[icolor] */
{
    /* Current algorithm: mean weighted by pixels (not colors) */
    /* Note it is important to get the rounding correct! */
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    hist3d histogram = cquantize->histogram;
    histptr histp;
    int c0, c1, c2;
    int c0min, c0max, c1min, c1max, c2min, c2max;
    long count;
    long total = 0;
    long c0total = 0;
    long c1total = 0;
    long c2total = 0;

    c0min = boxp->c0min; c0max = boxp->c0max;
    c1min = boxp->c1min; c1max = boxp->c1max;
    c2min = boxp->c2min; c2max = boxp->c2max;

    for (c0 = c0min; c0 <= c0max; c0++)
        for (c1 = c1min; c1 <= c1max; c1++) {
            histp = & histogram[c0][c1][c2min];
            for (c2 = c2min; c2 <= c2max; c2++) {
                if ((count = *histp++) != 0) {
                    total += count;
                    c0total += ((c0 << C0_SHIFT) + ((1<<C0_SHIFT)>>1)) * count;
                    c1total += ((c1 << C1_SHIFT) + ((1<<C1_SHIFT)>>1)) * count;
                    c2total += ((c2 << C2_SHIFT) + ((1<<C2_SHIFT)>>1)) * count;
                }
            }
        }

    cinfo->colormap[0][icolor] = (JSAMPLE) ((c0total + (total>>1)) / total);
    cinfo->colormap[1][icolor] = (JSAMPLE) ((c1total + (total>>1)) / total);
    cinfo->colormap[2][icolor] = (JSAMPLE) ((c2total + (total>>1)) / total);
}

```

```

LOCAL(void)
select_colors (j_decompress_ptr cinfo, int desired_colors)
/* Master routine for color selection */
{
    boxptr boxlist;
    int numboxes;
    int i;

    /* Allocate workspace for box list */
    boxlist = (boxptr) (*cinfo->mem->alloc_small)
        ((j_common_ptr) cinfo, JPOOL_IMAGE, desired_colors * SIZEOF(box));
    /* Initialize one box containing whole space */
    numboxes = 1;
    boxlist[0].c0min = 0;
    boxlist[0].c0max = MAXJSAMPLE >> C0_SHIFT;
    boxlist[0].c1min = 0;
    boxlist[0].c1max = MAXJSAMPLE >> C1_SHIFT;
    boxlist[0].c2min = 0;
    boxlist[0].c2max = MAXJSAMPLE >> C2_SHIFT;
    /* Shrink it to actually-used volume and set its statistics */
    update_box(cinfo, & boxlist[0]);
    /* Perform median-cut to produce final box list */
    numboxes = median_cut(cinfo, boxlist, numboxes, desired_colors);
}

```



```

/* Compute the representative color for each box, fill colormap */
for (i = 0; i < numboxes; i++)
    compute_color(cinfo, &boxlist[i], i);
cinfo->actual_number_of_colors = numboxes;
TRACEMS1(cinfo, 1, JERC_QUANT_SELECTED, numboxes);
}

```

```

/*
 * These routines are concerned with the time-critical task of mapping input
 * colors to the nearest color in the selected colormap.
 *
 * We re-use the histogram space as an "inverse color map", essentially a
 * cache for the results of nearest-color searches. All colors within a
 * histogram cell will be mapped to the same colormap entry, namely the one
 * closest to the cell's center. This may not be quite the closest entry to
 * the actual input color, but it's almost as good. A zero in the cache
 * indicates we haven't found the nearest color for that cell yet; the array
 * is cleared to zeroes before starting the mapping pass. When we find the
 * nearest color for a cell, its colormap index plus one is recorded in the
 * cache for future use. The pass2 scanning routines call fill_inverse_cmap
 * when they need to use an unfilled entry in the cache.
 *
 * Our method of efficiently finding nearest colors is based on the "locally
 * sorted search" idea described by Heckbert and on the incremental distance
 * calculation described by Spencer W. Thomas in chapter III.1 of Graphics
 * Gems II (James Arvo, ed. Academic Press, 1991). Thomas points out that
 * the distances from a given colormap entry to each cell of the histogram can
 * be computed quickly using an incremental method: the differences between
 * distances to adjacent cells themselves differ by a constant. This allows a
 * fairly fast implementation of the "brute force" approach of computing the
 * distance from every colormap entry to every histogram cell. Unfortunately,
 * it needs a work array to hold the best-distance-so-far for each histogram
 * cell (because the inner loop has to be over cells, not colormap entries).
 * The work array elements have to be INT32s, so the work array would need
 * 256Kb at our recommended precision. This is not feasible in DOS machines.
 *
 * To get around these problems, we apply Thomas' method to compute the
 * nearest colors for only the cells within a small subbox of the histogram.
 * The work array need be only as big as the subbox, so the memory usage
 * problem is solved. Furthermore, we need not fill subboxes that are never
 * referenced in pass2; many images use only part of the color gamut, so a
 * fair amount of work is saved. An additional advantage of this
 * approach is that we can apply Heckbert's locality criterion to quickly
 * eliminate colormap entries that are far away from the subbox; typically
 * three-fourths of the colormap entries are rejected by Heckbert's criterion,
 * and we need not compute their distances to individual cells in the subbox.
 * The speed of this approach is heavily influenced by the subbox size: too
 * small means too much overhead, too big loses because Heckbert's criterion
 * can't eliminate as many colormap entries. Empirically the best subbox
 * size seems to be about 1/512th of the histogram (1/8th in each direction).
 *
 * Thomas' article also describes a refined method which is asymptotically
 * faster than the brute-force method, but it is also far more complex and
 * cannot efficiently be applied to small subboxes. It is therefore not
 * useful for programs intended to be portable to DOS machines. On machines
 * with plenty of memory, filling the whole histogram in one shot with Thomas'
 * refined method might be faster than the present code --- but then again,
 * it might not be any faster, and it's certainly more complicated.
 */

```

```

/* log2(histogram cells in update box) for each axis; this can be adjusted */
#define BOX_C0_LOG (HIST_C0_BITS-3)
#define BOX_C1_LOG (HIST_C1_BITS-3)
#define BOX_C2_LOG (HIST_C2_BITS-3)

#define BOX_C0_ELEMS (1<<BOX_C0_LOG) /* # of hist cells in update box */
#define BOX_C1_ELEMS (1<<BOX_C1_LOG)
#define BOX_C2_ELEMS (1<<BOX_C2_LOG)

#define BOX_C0_SHIFT (C0_SHIFT + BOX_C0_LOG)
#define BOX_C1_SHIFT (C1_SHIFT + BOX_C1_LOG)
#define BOX_C2_SHIFT (C2_SHIFT + BOX_C2_LOG)

```

```

/*
 * The next three routines implement inverse colormap filling. They could
 * all be folded into one big routine, but splitting them up this way saves
 * some stack space (the mindist[] and bestdist[] arrays need not coexist)

```

```

* and may allow some compilers to produce better code by registerizing more
* inner-loop variables.
*/

```

```

LOCAL(int)
find_nearby_colors(j_recompress_ptr cinfo, int minc0, int minc1, int minc2,
                  JSAMPLE colorlist[])
/* Locate the colormap entries close enough to an update box to be candidates
 * for the nearest entry to some cell(s) in the update box. The update box
 * is specified by the center coordinates of its first cell. The number of
 * candidate colormap entries is returned, and their colormap indexes are
 * placed in colorlist[].
 * This routine uses Heckbert's "locally sorted search" criterion to select
 * the colors that need further consideration.
 */
{
    int numcolors = cinfo->actual_number_of_colors;
    int maxc0, maxc1, maxc2;
    int centerc0, centerc1, centerc2;
    int i, x, ncolors;
    INT32 minmaxdist, min_dist, max_dist, tdist;
    INT32 mindist[MAXNUMCOLORS]; /* min distance to colormap entry i */

    /* Compute true coordinates of update box's upper corner and center.
     * Actually we compute the coordinates of the center of the upper-corner
     * histogram cell, which are the upper bounds of the volume we care about.
     * Note that since ">>" rounds down, the "center" values may be closer to
     * min than to max; hence comparisons to them must be "<=", not "<".
     */
    maxc0 = minc0 + ((1 << BOX_C0_SHIFT) - (1 << C0_SHIFT));
    centerc0 = (minc0 + maxc0) >> 1;
    maxc1 = minc1 + ((1 << BOX_C1_SHIFT) - (1 << C1_SHIFT));
    centerc1 = (minc1 + maxc1) >> 1;
    maxc2 = minc2 + ((1 << BOX_C2_SHIFT) - (1 << C2_SHIFT));
    centerc2 = (minc2 + maxc2) >> 1;

    /* For each color in colormap, find:
     * 1. its minimum squared-distance to any point in the update box
     *    (zero if color is within update box);
     * 2. its maximum squared-distance to any point in the update box.
     * Both of these can be found by considering only the corners of the box.
     * We save the minimum distance for each color in mindist[];
     * only the smallest maximum distance is of interest.
     */
    minmaxdist = 0x7FFFFFFF;

    for (i = 0; i < numcolors; i++) {
        /* We compute the squared-c0-distance term, then add in the other two. */
        x = GETJSAMPLE(cinfo->colormap[0][i]);
        if (x < minc0) {
            tdist = (x - minc0) * C0_SCALE;
            min_dist = tdist*tdist;
            tdist = (x - maxc0) * C0_SCALE;
            max_dist = tdist*tdist;
        } else if (x > maxc0) {
            tdist = (x - maxc0) * C0_SCALE;
            min_dist = tdist*tdist;
            tdist = (x - minc0) * C0_SCALE;
            max_dist = tdist*tdist;
        } else {
            /* within cell range so no contribution to min_dist */
            min_dist = 0;
            if (x <= centerc0) {
                tdist = (x - maxc0) * C0_SCALE;
                max_dist = tdist*tdist;
            } else {
                tdist = (x - minc0) * C0_SCALE;
                max_dist = tdist*tdist;
            }
        }

        x = GETJSAMPLE(cinfo->colormap[1][i]);
        if (x < minc1) {
            tdist = (x - minc1) * C1_SCALE;
            min_dist += tdist*tdist;
            tdist = (x - maxc1) * C1_SCALE;
            max_dist += tdist*tdist;
        } else if (x > maxc1) {
            tdist = (x - maxc1) * C1_SCALE;
            min_dist += tdist*tdist;
        }
    }
}

```

```

    tdist = (x - minc1) * C1_SCALE;
    max_dist += tdist*tdist;
} else {
    /* within cell range so no contribution to min_dist */
    if (x <= center1) {
        tdist = (x - maxc1) * C1_SCALE;
        max_dist += tdist*tdist;
    } else {
        tdist = (x - minc1) * C1_SCALE;
        max_dist += tdist*tdist;
    }
}

x = GETJSAMPLE(cinfo->colormap[2][i]);
if (x < minc2) {
    tdist = (x - minc2) * C2_SCALE;
    min_dist += tdist*tdist;
    tdist = (x - maxc2) * C2_SCALE;
    max_dist += tdist*tdist;
} else if (x > maxc2) {
    tdist = (x - maxc2) * C2_SCALE;
    min_dist += tdist*tdist;
    tdist = (x - minc2) * C2_SCALE;
    max_dist += tdist*tdist;
} else {
    /* within cell range so no contribution to min_dist */
    if (x <= center2) {
        tdist = (x - maxc2) * C2_SCALE;
        max_dist += tdist*tdist;
    } else {
        tdist = (x - minc2) * C2_SCALE;
        max_dist += tdist*tdist;
    }
}

mindist[i] = min_dist; /* save away the results */
if (max_dist < minmaxdist)
    minmaxdist = max_dist;
}

/* Now we know that no cell in the update box is more than minmaxdist
 * away from some colormap entry. Therefore, only colors that are
 * within minmaxdist of some part of the box need be considered.
 */
ncolors = 0;
for (i = 0; i < numcolors; i++) {
    if (mindist[i] <= minmaxdist)
        colorlist[ncolors++] = (JSAMPLE) i;
}
return ncolors;
}

LOCAL(void)
find_best_colors(j_decompress_ptr cinfo, int minc0, int minc1, int minc2,
                int numcolors, JSAMPLE colorlist[], JSAMPLE bestcolor[])
/* Find the closest colormap entry for each cell in the update box,
 * given the list of candidate colors prepared by find_nearby_colors.
 * Return the indexes of the closest entries in the bestcolor[] array.
 * This routine uses Thomas' incremental distance calculation method to
 * find the distance from a colormap entry to successive cells in the box.
 */
{
    int ic0, ic1, ic2;
    int i, icolor;
    register INT32 * bptr; /* pointer into bestdist[] array */
    JSAMPLE * cptr; /* pointer into bestcolor[] array */
    INT32 dist0, dist1; /* initial distance values */
    register INT32 dist2; /* current distance in inner loop */
    INT32 xx0, xx1; /* distance increments */
    register INT32 xx2;
    INT32 inc0, inc1, inc2; /* initial values for increments */
    /* This array holds the distance to the nearest-so-far color for each cell */
    INT32 bestdist[BOX_C0_ELEMS * BOX_C1_ELEMS * BOX_C2_ELEMS];

    /* Initialize best-distance for each cell of the update box */
    bptr = bestdist;
    for (i = BOX_C0_ELEMS*BOX_C1_ELEMS*BOX_C2_ELEMS-1; i >= 0; i--)
        *bptr++ = 0x7FFFFFFF;

```

```

/* For each color selected by find_nearby_colors,
 * compute its distance to the center of each cell in the box.
 * If that's less than best-so-far, update best distance and color number.
 */

/* Nominal steps between cell centers ("x" in Thomas article) */
#define STEP_C0 ((1 << C0_SHIFT) * C0_SCALE)
#define STEP_C1 ((1 << C1_SHIFT) * C1_SCALE)
#define STEP_C2 ((1 << C2_SHIFT) * C2_SCALE)

for (i = 0; i < numcolors; i++) {
    icolor = GETJSAMPLE(colorlist[i]);
    /* Compute (square of) distance from minc0/c1/c2 to this color */
    inc0 = (minc0 - GETJSAMPLE(cinfo->colormap[0][icolor])) * C0_SCALE;
    dist0 = inc0*inc0;
    inc1 = (minc1 - GETJSAMPLE(cinfo->colormap[1][icolor])) * C1_SCALE;
    dist0 += inc1*inc1;
    inc2 = (minc2 - GETJSAMPLE(cinfo->colormap[2][icolor])) * C2_SCALE;
    dist0 += inc2*inc2;
    /* Form the initial difference increments */
    inc0 = inc0 * (2 * STEP_C0) + STEP_C0 * STEP_C0;
    inc1 = inc1 * (2 * STEP_C1) + STEP_C1 * STEP_C1;
    inc2 = inc2 * (2 * STEP_C2) + STEP_C2 * STEP_C2;
    /* Now loop over all cells in box, updating distance per Thomas method */
    bptr = bestdist;
    cptr = bestcolor;
    xx0 = inc0;
    for (ic0 = BOX_C0_ELEMS-1; ic0 >= 0; ic0--) {
        dist1 = dist0;
        xx1 = inc1;
        for (ic1 = BOX_C1_ELEMS-1; ic1 >= 0; ic1--) {
            dist2 = dist1;
            xx2 = inc2;
            for (ic2 = BOX_C2_ELEMS-1; ic2 >= 0; ic2--) {
                if (dist2 < *bptr) {
                    *bptr = dist2;
                    *cptr = (JSAMPLE) icolor;
                }
                dist2 += xx2;
                xx2 += 2 * STEP_C2 * STEP_C2;
                bptr++;
                cptr++;
            }
            dist1 += xx1;
            xx1 += 2 * STEP_C1 * STEP_C1;
        }
        dist0 += xx0;
        xx0 += 2 * STEP_C0 * STEP_C0;
    }
}

LOCAL(void)
fill_inverse_cmap (j_decompress_ptr cinfo, int c0, int c1, int c2)
/* Fill the inverse colormap entries in the update box that contains */
/* histogram cell c0/c1/c2. (Only that one cell MUST be filled, but */
/* we can fill as many others as we wish.) */
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    hist3d histogram = cquantize->histogram;
    int minc0, minc1, minc2; /* lower left corner of update box */
    int ic0, ic1, ic2;
    register JSAMPLE * cptr; /* pointer into bestcolor[] array */
    register histptr cachep; /* pointer into main cache array */
    /* This array lists the candidate colormap indexes. */
    JSAMPLE colorlist[MAXNUMCOLORS];
    int numcolors; /* number of candidate colors */
    /* This array holds the actually closest colormap index for each cell. */
    JSAMPLE bestcolor[BOX_C0_ELEMS * BOX_C1_ELEMS * BOX_C2_ELEMS];

    /* Convert cell coordinates to update box ID */
    c0 >>= BOX_C0_LOG;
    c1 >>= BOX_C1_LOG;
    c2 >>= BOX_C2_LOG;

    /* Compute true coordinates of update box's origin corner.
     * Actually we compute the coordinates of the center of the corner
     * histogram cell, which are the lower bounds of the volume we care about.
     */

```

```

minc0 = (c0 << BOX_C0_SHIFT) + ((1 << C0_SHIFT) >> 1);
minc1 = (c1 << BOX_C1_SHIFT) + ((1 << C1_SHIFT) >> 1);
minc2 = (c2 << BOX_C2_SHIFT) + ((1 << C2_SHIFT) >> 1);

/* Determine which colormap entries are close enough to be candidates
 * for the nearest entry to some cell in the update box.
 */
numcolors = find_nearby_colors(cinfo, minc0, minc1, minc2, colorlist);

/* Determine the actually nearest colors. */
find_best_colors(cinfo, minc0, minc1, minc2, numcolors, colorlist,
    bestcolor);

/* Save the best color numbers (plus 1) in the main cache array */
c0 <=<= BOX_C0_LOG; /* convert ID back to base cell indexes */
c1 <=<= BOX_C1_LOG;
c2 <=<= BOX_C2_LOG;
cptr = bestcolor;
for (ic0 = 0; ic0 < BOX_C0_ELEMS; ic0++) {
    for (ic1 = 0; ic1 < BOX_C1_ELEMS; ic1++) {
        cachep = & histogram[c0+ic0][c1+ic1][c2];
        for (ic2 = 0; ic2 < BOX_C2_ELEMS; ic2++) {
            *cachep++ = (histcell) (GETJSAMPLE(*cptr++) + 1);
        }
    }
}

/*
 * Map some rows of pixels to the output colormapped representation.
 */
METHODDEF(void)
pass2_no_dither (j_decompress_ptr cinfo,
    JSAMPARRAY input_buf, JSAMPARRAY output_buf, int num_rows)
/* This version performs no dithering */
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    hist3d histogram = cquantize->histogram;
    register JSAMPROW inptr, outptr;
    register histptr cachep;
    register int c0, c1, c2;
    int row;
    JDIMENSION col;
    JDIMENSION width = cinfo->output_width;

    for (row = 0; row < num_rows; row++) {
        inptr = input_buf[row];
        outptr = output_buf[row];
        for (col = width; col > 0; col--) {
            /* get pixel value and index into the cache */
            c0 = GETJSAMPLE(*inptr++) >> C0_SHIFT;
            c1 = GETJSAMPLE(*inptr++) >> C1_SHIFT;
            c2 = GETJSAMPLE(*inptr++) >> C2_SHIFT;
            cachep = & histogram[c0][c1][c2];
            /* If we have not seen this color before, find nearest colormap entry */
            /* and update the cache */
            if (*cachep == 0)
                fill_inverse_cmap(cinfo, c0, c1, c2);
            /* Now emit the colormap index for this cell */
            *outptr++ = (JSAMPLE) (*cachep - 1);
        }
    }
}

METHODDEF(void)
pass2_fs_dither (j_decompress_ptr cinfo,
    JSAMPARRAY input_buf, JSAMPARRAY output_buf, int num_rows)
/* This version performs Floyd-Steinberg dithering */
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    hist3d histogram = cquantize->histogram;
    register LOCFSEERROR cur0, cur1, cur2; /* current error or pixel value */
    register belowerr0, belowerr1, belowerr2; /* error for pixel below cur */
    register bpreverr0, bpreverr1, bpreverr2; /* error for below/prev col */
    register FSERRPTR errorptr; /* => fseerrors[] at column before current */
    JSAMPROW inptr; /* => current input pixel */
    JSAMPROW outptr; /* => current output pixel */

```

```

histptr cachep;
int dir;          /* +1 or -1 depending on direction */
int dir3;         /* 3*dir, for advancing inptr & errorptr */
int row;
JDIMENSION col;
JDIMENSION width = cinfo->output_width;
JSAMPLE *range_limit = cinfo->sample_range_limit;
int *error_limit = cquantize->error_limiter;
JSAMPROW colormap0 = cinfo->colormap[0];
JSAMPROW colormap1 = cinfo->colormap[1];
JSAMPROW colormap2 = cinfo->colormap[2];
SHIFT_TEMPS

for (row = 0; row < num_rows; row++) {
    inptr = input_buf[row];
    outptr = output_buf[row];
    if (cquantize->on_odd_row) {
        /* work right to left in this row */
        inptr += (width-1) * 3; /* so point to rightmost pixel */
        outptr += width-1;
        dir = -1;
        dir3 = -3;
        errorptr = cquantize->ferrors + (width+1)*3; /* => entry after last column */
        cquantize->on_odd_row = FALSE; /* flip for next time */
    } else {
        /* work left to right in this row */
        dir = 1;
        dir3 = 3;
        errorptr = cquantize->ferrors; /* => entry before first real column */
        cquantize->on_odd_row = TRUE; /* flip for next time */
    }
    /* Preset error values: no error propagated to first pixel from left */
    cur0 = cur1 = cur2 = 0;
    /* and no error propagated to row below yet */
    belowerr0 = belowerr1 = belowerr2 = 0;
    bpreverr3 = bpreverr1 = bpreverr2 = 0;

    for (col = width; col > 0; col--) {
        /* curN holds the error propagated from the previous pixel on the
         * current line. Add the error propagated from the previous line
         * to form the complete error correction term for this pixel, and
         * round the error term (which is expressed * 16) to an integer.
         * RIGHT_SHIFT rounds towards minus infinity, so adding 8 is correct
         * for either sign of the error value.
         * Note: errorptr points to *previous* column's array entry.
         */
        cur0 = RIGHT_SHIFT(cur0 + errorptr[dir3+0] + 8, 4);
        cur1 = RIGHT_SHIFT(cur1 + errorptr[dir3+1] + 8, 4);
        cur2 = RIGHT_SHIFT(cur2 + errorptr[dir3+2] + 8, 4);
        /* Limit the error using transfer function set by init_error_limit.
         * See comments with init_error_limit for rationale.
         */
        cur0 = error_limit[cur0];
        cur1 = error_limit[cur1];
        cur2 = error_limit[cur2];
        /* Form pixel value + error, and range-limit to 0..MAXJSAMPLE.
         * The maximum error is +- MAXJSAMPLE (or less with error limiting);
         * this sets the required size of the range_limit array.
         */
        cur0 += GETJSAMPLE(inptr[0]);
        cur1 += GETJSAMPLE(inptr[1]);
        cur2 += GETJSAMPLE(inptr[2]);
        cur0 = GETJSAMPLE(range_limit[cur0]);
        cur1 = GETJSAMPLE(range_limit[cur1]);
        cur2 = GETJSAMPLE(range_limit[cur2]);
        /* Index into the cache with adjusted pixel value */
        cachep = & histogram[cur0>>C0_SHIFT][cur1>>C1_SHIFT][cur2>>C2_SHIFT];
        /* If we have not seen this color before, find nearest colormap */
        /* entry and update the cache */
        if (*cachep == 0)
            fill_inverse_cmap(cinfo, cur0>>C0_SHIFT, cur1>>C1_SHIFT, cur2>>C2_SHIFT);
        /* Now emit the colormap index for this cell */
        { register int pixcode = *cachep - 1;
          *outptr = (JSAMPLE) pixcode;
          /* Compute representation error for this pixel */
          cur0 -= GETJSAMPLE(colormap0[pixcode]);
          cur1 -= GETJSAMPLE(colormap1[pixcode]);
          cur2 -= GETJSAMPLE(colormap2[pixcode]);
        }
        /* Compute error fractions to be propagated to adjacent pixels.

```

```

    * Add these into the running sums, and simultaneously shift the
    * next-line error sums left by 1 column.
    */
    { register LOCFSEERROR bnexterr, delta;

    bnexterr = cur0;      /* Process component 0 */
    delta = cur0 * 2;
    cur0 += delta;        /* form error * 3 */
    errorptr[0] = (FSEPROR) (bpreverr0 + cur0);
    cur0 += delta;        /* form error * 5 */
    bpreverr0 = belowerr0 + cur0;
    belowerr0 = bnexterr;
    cur0 -= delta;        /* form error * 7 */
    bnexterr = cur1;      /* Process component 1 */
    delta = cur1 * 2;
    cur1 += delta;        /* form error * 3 */
    errorptr[1] = (FSEERROR) (bpreverr1 + cur1);
    cur1 += delta;        /* form error * 5 */
    bpreverr1 = belowerr1 + cur1;
    belowerr1 = bnexterr;
    cur1 += delta;        /* form error * 7 */
    bnexterr = cur2;      /* Process component 2 */
    delta = cur2 * 2;
    cur2 += delta;        /* form error * 3 */
    errorptr[2] = (FSEERROR) (bpreverr2 + cur2);
    cur2 += delta;        /* form error * 5 */
    bpreverr2 = belowerr2 + cur2;
    belowerr2 = bnexterr;
    cur2 += delta;        /* form error * 7 */
    }
    /* At this point curN contains the 7/16 error value to be propagated
    * to the next pixel on the current line, and all the errors for the
    * next line have been shifted over. We are therefore ready to move on.
    */
    inptr += dir3;        /* Advance pixel pointers to next column */
    ouptr += dir;
    errorptr += dir3;     /* advance errorptr to current column */
}
/* Post-loop cleanup: we must unload the final error values into the
 * final fserrors[] entry. Note we need not unload belowerrN because
 * it is for the dummy column before or after the actual array.
 */
errorptr[0] = (FSEERROR) bpreverr0; /* unload prev errs into array */
errorptr[1] = (FSEERROR) bpreverr1;
errorptr[2] = (FSEERROR) bpreverr2;

Initialize the error-limiting transfer function (lookup table).
* The raw F-S error computation can potentially compute error values of up to
* +- MAXJSAMPLE. But we want the maximum correction applied to a pixel to be
* much less, otherwise obviously wrong pixels will be created. (Typical
* effects include weird fringes at color-area boundaries, isolated bright
* pixels in a dark area, etc.) The standard advice for avoiding this problem
* is to ensure that the "corners" of the color cube are allocated as output
* colors; then repeated errors in the same direction cannot cause cascading
* error buildup. However, that only prevents the error from getting
* completely out of hand; Aaron Giles reports that error limiting improves
* the results even with corner colors allocated.
* A simple clamping of the error values to about +- MAXJSAMPLE/8 works pretty
* well, but the smoother transfer function used below is even better. Thanks
* to Aaron Giles for this idea.
*/

LOCAL(void)
init_error_limit (j_decompress_ptr cinfo)
/* Allocate and fill in the error_limiter table */
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    int *table;
    int in, out;

    table = (int *) (*cinfo->mem->alloc_small)
        ((j_common_ptr) cinfo, JPOOL_IMAGE, (MAXJSAMPLE*2+1) * sizeof(int));
    table += MAXJSAMPLE; /* so can index -MAXJSAMPLE .. +MAXJSAMPLE */
    cquantize->error_limiter = table;

#define STEPSTZ ((MAXJSAMPLE+1)/16)

```

```

/* Map errors 1:1 up to +- MAXJSAMPLE/16 */
out = 0;
for (in = 0, in < STEPSIZE; in++, out++) {
    table[in] = out; table[-in] = -out;
}
/* Map errors 1:2 up to +- 3*MAXJSAMPLE/16 */
for (; in < STEPSIZE*3; in++, out += (in&1) ? 0 : 1) {
    table[in] = out; table[-in] = -out;
}
/* Clamp the rest to final out value (which is (MAXJSAMPLE+1)/8) */
for (; in <= MAXJSAMPLE; in++) {
    table[in] = out; table[-in] = -out;
}
#undef STEPSIZE
}

/*
 * Finish up at the end of each pass.
 */

METHODDEF(void)
finish_pass1 (j_decompress_ptr cinfo)
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;

    /* Select the representative colors and fill in cinfo->colormap */
    cinfo->colormap = cquantize->sv_colormap;
    select_colors(cinfo, cquantize->desired);
    /* Force next pass to zero the color index table */
    cquantize->needs_zeroed = TRUE;
}

METHODDEF(void)
finish_pass2 (j_decompress_ptr cinfo)
{
    /* no work */
}

/*
 * Initialize for each processing pass.
 */

METHODDEF(void)
start_pass_2_quant (j_decompress_ptr cinfo, boolean is_pre_scan)
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;
    hist3d histogram = cquantize->histogram;
    int i;

    /* Only F-S dithering or no dithering is supported. */
    /* If user asks for ordered dither, give him F-S. */
    if (cinfo->dither_mode != JDITHER_NONE)
        cinfo->dither_mode = JDITHER_FS;

    if (is_pre_scan) {
        /* Set up method pointers */
        cquantize->pub.color_quantize = prescan_quantize;
        cquantize->pub.finish_pass = finish_pass1;
        cquantize->needs_zeroed = TRUE; /* Always zero histogram */
    } else {
        /* Set up method pointers */
        if (cinfo->dither_mode == JDITHER_FS)
            cquantize->pub.color_quantize = pass2_fs_dither;
        else
            cquantize->pub.color_quantize = pass2_no_dither;
        cquantize->pub.finish_pass = finish_pass2;

        /* Make sure color count is acceptable */
        i = cinfo->actual_number_of_colors;
        if (i < 1)
            ERREXIT1(cinfo, JERR_QUANT_FEW_COLORS, 1);
        if (i > MAXNUMCOLORS)
            ERREXIT1(cinfo, JERR_QUANT_MANY_COLORS, MAXNUMCOLORS);

        if (cinfo->dither_mode == JDITHER_FS) {
            size_t arraysize = (size_t) ((cinfo->output_width + 2) *
                                         (3 * sizeof(FSERROR)));

```



```

/* Allocate Floyd-Steinberg workspace if we didn't already. */
if (cquantize->fserrors == NULL)
    cquantize->fserrors = (FSERRPTR) (*cinfo->mem->alloc_large)
        ((j_common_ptr) cinfo, JPOOL_IMAGE, arraysizes);
/* Initialize the propagated errors to zero. */
jzero_far((void *) cquantize->fserrors, arraysizes);
/* Make the error-limit table if we didn't already. */
if (cquantize->error_limiter == NULL)
    init_error_limit(cinfo);
cquantize->on_odd_row = FALSE;
}

/* Zero the histogram or inverse color map, if necessary */
if (cquantize->needs_zeroed) {
    for (i = 0; i < HIST_C0_ELEMS; i++) {
        jzero_far((void *) histogram[i],
            HIST_C1_ELEMS*HIST_C2_ELEMS * SIZEOF(histcell));
    }
    cquantize->needs_zeroed = FALSE;
}

/*
 * Switch to a new external colormap between output passes.
 */

METHODDEF(void)
new_color_map_2_quant (j_decompress_ptr cinfo)
{
    my_cquantize_ptr cquantize = (my_cquantize_ptr) cinfo->cquantize;

    /* Reset the inverse color map */
    cquantize->needs_zeroed = TRUE;

    /*
     * Module initialization routine for 2-pass color quantization.
     */
    GLOBAL(void)
    jinit_2pass_quantizer (j_decompress_ptr cinfo)
    {
        my_cquantize_ptr cquantize;
        int i;

        cquantize = (my_cquantize_ptr)
            (*cinfo->mem->alloc_small) ((j_common_ptr) cinfo, JPOOL_IMAGE,
                SIZEOF(my_cquantizer));
        cinfo->cquantize = (struct jpeg_color_quantizer *) cquantize;
        cquantize->pub.start_pass = start_pass_2_quant;
        cquantize->pub.new_color_map = new_color_map_2_quant;
        cquantize->fserrors = NULL; /* flag optional arrays not allocated */
        cquantize->error_limiter = NULL;

        /* Make sure jdmaster didn't give me a case I can't handle */
        if ((cinfo->out_color_components != 3)
            || ERREXIT(cinfo, JERR_NOTIMPL);

        /* Allocate the histogram/inverse colormap storage */
        cquantize->histogram = (hist3d) (*cinfo->mem->alloc_small)
            ((j_common_ptr) cinfo, JPOOL_IMAGE, HIST_C0_ELEMS * SIZEOF(hist2d));
        for (i = 0; i < HIST_C0_ELEMS; i++) {
            cquantize->histogram[i] = (hist2d) (*cinfo->mem->alloc_large)
                ((j_common_ptr) cinfo, JPOOL_IMAGE,
                    HIST_C1_ELEMS*HIST_C2_ELEMS * SIZEOF(histcell));
        }
        cquantize->needs_zeroed = TRUE; /* histogram is garbage now */

        /* Allocate storage for the completed colormap, if required.
         * We do this now since it is FAR storage and may affect
         * the memory manager's space calculations.
         */
        if (cinfo->enable_2pass_quant) {
            /* Make sure color count is acceptable */
            int desired = cinfo->desired_number_of_colors;
            /* Lower bound on # of colors ... somewhat arbitrary as long as > 0 */
            if (desired < 8)

```

```

    ERREXT1(cinfo, JERR_QUANT_FEW_COLORS, 8);
    /* Make sure colormap indexes can be represented by JSAMPLEs */
    if (desired > MAXNUMCOLORS)
        ERREXT1(cinfo, JERR_QUANT_MANY_COLORS, MAXNUMCOLORS);
    cquantize->sv_colormap = (*cinfo->mem->alloc_sarray)
        ((j_common_ptr) cinfo, JPOOL_IMAGE, (JDIMENSION) desired, (JDIMENSION) 3);
    cquantize->desired = desired;
} else
    cquantize->sv_colormap = NULL;

/* Only F-S dithering or no dithering is supported. */
/* If user asks for ordered dither, give him F-S. */
if (cinfo->dither_mode != JDITHER_NONE)
    cinfo->dither_mode = JDITHER_FS;

/* Allocate Floyd-Steinberg workspace if necessary.
 * This isn't really needed until pass 2, but again it is FAR storage.
 * Although we will cope with a later change in dither_mode,
 * we do not promise to honor max_memory_to_use if dither_mode changes.
 */
if (cinfo->dither_mode == JDITHER_FS) {
    cquantize->fserrors = (FSERRPTR) (*cinfo->mem->alloc_large)
        ((j_common_ptr) cinfo, JPOOL_IMAGE,
         (size_t) ((cinfo->output_width + 2) * (3 * sizeof(FSERROR))));
    /* Might as well create the error-limiting table too. */
    init_error_limit(cinfo);
}
}

#endif /* QUANT_2PASS_SUPPORTED */

```

```

/*
 * jutil.c
 *
 * Copyright (C) 1991-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains tables and miscellaneous utility routines needed
 * for both compression and decompression.
 * Note we prefix all global names with "j" to minimize conflicts with
 * a surrounding application.
 */

```

```

#define JPEG_INTERNALS
#include "jinclude.h"
#include "jpeglib.h"

```

```

/*
 * jpeg_zigzag_order[i] is the zigzag-order position of the i'th element
 * of a DCT block read in natural order (left to right, top to bottom).
 */

```

```

#if 0 /* This table is not actually needed in v6a */

```

```

const int jpeg_zigzag_order[DCTSIZE2] = {
  0, 1, 5, 6, 14, 15, 27, 28,
  2, 4, 7, 13, 16, 26, 29, 42,
  3, 8, 12, 17, 25, 30, 41, 43,
  9, 11, 18, 24, 31, 40, 44, 53,
  10, 19, 23, 32, 39, 45, 52, 54,
  20, 22, 33, 38, 46, 51, 55, 60,
  21, 34, 37, 47, 50, 56, 59, 61,
  35, 36, 48, 49, 57, 58, 62, 63

```

```

#endif

```

```

/*
 * jpeg_natural_order[i] is the natural-order position of the i'th element
 * of zigzag order.

```

```

 * When reading corrupted data, the Huffman decoders could attempt
 * to reference an entry beyond the end of this array (if the decoded
 * zero run length reaches past the end of the block). To prevent
 * wild stores without adding an inner-loop test, we put some extra
 * "63"s after the real entries. This will cause the extra coefficient
 * to be stored in location 63 of the block, not somewhere random.
 * The worst case would be a run-length of 15, which means we need 16
 * fake entries.
 */

```

```

const int jpeg_natural_order[DCTSIZE2+16] = {
  0, 1, 8, 16, 9, 2, 3, 10,
  17, 24, 32, 25, 18, 11, 4, 5,
  12, 19, 26, 33, 40, 48, 41, 34,
  27, 20, 13, 6, 7, 14, 21, 28,
  35, 42, 49, 56, 57, 50, 43, 36,
  29, 22, 15, 23, 30, 37, 44, 51,
  58, 59, 32, 45, 38, 31, 39, 46,
  53, 60, 61, 54, 47, 55, 62, 63,
  63, 63, 63, 63, 63, 63, 63, 63, /* extra entries for safety in decoder */
  63, 63, 63, 63, 63, 63, 63, 63
};

```

```

/*
 * Arithmetic utilities
 */

```

```

GLOBAL(long)
jdiv_round_up(long a, long b)
/* Compute a/b rounded up to next integer, ie, ceil(a/b) */
/* Assumes a >= 0, b > 0 */
{
  return (a + b - 1L) / b;
}

```

```

GLOBAL(long)

```

```

jround_up (long a, long b)
/* Compute a rounded up to next multiple of b, ie, ceil(a/b)*b */
/* Assumed a >= 0, b > 0 */
{
    a += b - 1;
    return a - (a % b);
}

/* On normal machines we can apply MEMCOPY() and MEMZERO() to sample arrays
* and coefficient-block arrays. This won't work on 80x86 because the arrays
* are FAR and we're assuming a small-pointer memory model. However, some
* DOS compilers provide far-pointer versions of memcpy() and memset() even
* in the small-model libraries. These will be used if USE_FMEM is defined.
* Otherwise, the routines below do it the hard way. (The performance cost
* is not all that great, because these routines aren't very heavily used.)
*/

#ifdef NEED_FAR_POINTERS /* normal case, same as regular macros */
#define FMEMCOPY(dest,src,size) MEMCOPY(dest,src,size)
#define FMEMZERO(target,size) MEMZERO(target,size)
#else /* 80x86 case, define if we can */
#ifdef USE_FMEM
#define FMEMCOPY(dest,src,size) _fmemcpy((void *) (dest), (const void *) (src), (size_t) (size))
#define FMEMZERO(target,size) _fmemset((void *) (target), 0, (size_t) (size))
#else
#endif
#endif

GLOBAL(void)
jcopy_sample_rows (JSAMPARRAY input_array, int source_row,
                   JSAMPARRAY output_array, int dest_row,
                   int num_rows, JDIMENSION num_cols)
/* Copy num_rows rows of samples from one place to another.
num_rows rows are copied from input_array[source_row++]
to output_array[dest_row++]; these areas may overlap for duplication.
The source and destination arrays must be at least as wide as num_cols.
*/
{
    register JSAMPROW inptr, outptr;
#ifdef FMEMCOPY
    register size_t count = (size_t) (num_cols * SIZEOF(JSAMPLE));
#else
    register JDIMENSION count;
#endif
    register int row;

    input_array += source_row;
    output_array += dest_row;

    for (row = num_rows; row > 0; row--) {
        inptr = *input_array++;
        outptr = *output_array++;
#ifdef FMEMCOPY
        FMEMCOPY(outptr, inptr, count);
#else
        for (count = num_cols; count > 0; count--)
            *outptr++ = *inptr++; /* needn't bother with GETJSAMPLE() here */
#endif
    }
}

GLOBAL(void)
jcopy_block_row (JBLOCKROW input_row, JBLOCKROW output_row,
                 JDIMENSION num_blocks)
/* Copy num_blocks of coefficient blocks from one place to another. */
{
#ifdef FMEMCOPY
    FMEMCOPY(output_row, input_row, num_blocks * (DCTSIZE2 * SIZEOF(JCOEF)));
#else
    register JCOEFPTR inptr, outptr;
    register long count;

    inptr = (JCOEFPTR) input_row;
    outptr = (JCOEFPTR) output_row;
    for (count = (long) num_blocks * DCTSIZE2; count > 0; count--) {
        *outptr++ = *inptr++;
    }
#endif
}

```

}

GLOBAL (void)

```
jzero_fam (word * target, size_t bytestozero)
```

```
/* Zero out a chunk of FAR memory. */
```

```
/* This might be sample-array data, block-array data, or alloc_large data. */
```

{

```
#ifndef FIL_GML-RO
```

```
FMEMZERO(target, bytestozero);
```

```
#else
```

```
register char * ptr = (char *) target;
```

```
register : test count;
```

```
for (count = bytestozero; count > 0; count--) {
```

```
*ptr = 0;
```

}

```
#endif
```

}

[illegible]

```

/*
 * @author: Imtiaz Hossain
 *
 * @version: 2.0
 *
 */

#include <stdio.h>
#include <malloc.h>
#include "jerror.h"
#include "setjmp.h"
#include "jpeg_class.h"

// static consts
// Error Definitions
const int JPEG::SUCCESS=0;
const int JPEG::MEMORY_ALLOC_ERROR=1;
const int JPEG::FILE_READ_ERROR=2;
const int JPEG::FILE_WRITE_ERROR=3;
const int JPEG::JPEG_LIB_STRUCT_INIT_ERROR=4;

// Coding types

const int JPEG::DEFAULT_CODING=0;
const int JPEG::BASELINE=0;
const int JPEG::PROGRESSIVE=1;
const int JPEG::LOSSLESS=2;

// Resolution
const int JPEG::DEFAULT_RES=1;
const int JPEG::ONE_BYTE=1;
const int JPEG::TWO_BYTE=2;
const int JPEG::THREE_BYTE=3;
const int JPEG::FOUR_BYTE=4;

// Color Spaces
const int JPEG::DEFAULT_BPP=3;
const int JPEG::Gray=1;
const int JPEG::RGB=3;

// Image Quality
const int JPEG::DEF_QUALITY=60;

// Compression Ratio
const float JPEG::DEF_RATIO=0.50;

// Time limit on the compression
const long JPEG::TIME=5; // 5 seconds .

BYTE * JPEG::Encode(BYTE *jpeg_get_Buffer, int *length, int *ret_quality, int n_Height, int n_Width, int n_Bpp, int n_Quality, int n_Res, int n_Coding)
{
    struct jpeg_compress_struct c_struct;
    struct jpeg_error_mgr jerr;
    JSAMPLE * ptr_to_buffer[1];
    int buffer_length, counter=0, i;
    JOCTET * returnbuffer;

    BYTE * jpeg_RGB_Buffer;
    int bpps;
    int res;

    bpps=n_Bpp;
    res=n_Res;

    jpeg_RGB_Buffer=Resolution_Convertor(jpeg_get_Buffer, &bpps, &res, n_Height, &n_Width);
    n_Bpp=bpps;

    n_Image_Height=n_Height;
    n_Image_Width=n_Width;
    n_Image_Bpp=n_Bpp;

```

```

//*** Need to assert BPP=1 or 3, and Width and Height >0
n_ImageResolution=n_Res;
n_ImageCodingType=n_Coding;
n_ImageQuality=n_Quality;

buffer_length=n_ImageWidth*n_ImageBpp;

ptr_to_buffer[0]=(JSAMPLE *)malloc(buffer_length*sizeof(JSAMPLE));

if(ptr_to_buffer[0]==NULL)
{
    fprintf(stderr,"Memory Allocation error!!!\n");
    exit(1);
}

// First Step : Initializing JPEG compression object
// jpeglib : error handling code
c_struct.err = jpeg_std_error(&jerr);

jpeg_create_compress(&c_struct); // object initialization
c_struct.index=0;

// Next Step (2) : We will not be using a File. So skipping source definition.
//               Initializing Height, Width and Bpp properties.
c_struct.image_width = n_ImageWidth;
c_struct.image_height = n_ImageHeight;
if (n_ImageBpp==3)
{
    c_struct.input_components = 3; // # of color components/Bytes per pixel.
    c_struct.in_color_space = JCS_RGB; // colorspace for input image.
}
else
{
    c_struct.input_components = 1; // # of color components/Bytes per pixel.
    c_struct.in_color_space = JCS_GRAYSCALE; // colorspace for input image.
}

jpeg_set_dest(&c_struct); // ### new modification. Instead of jpeg_stdio_src.

// Next Step (3) : Setting defaults and any special parameters like Quality, Coding, etc..
jpeg_set_defaults(&c_struct);
jpeg_set_quality(&c_struct, n_ImageQuality, TRUE);

// Next Step (4) : Initializing compressor
jpeg_start_compress(&c_struct, TRUE);

// Next Step (5) : Compressing one scanline at a time.
while(c_struct.next_scanline < c_struct.image_height)
{
    for(i=0;i<buffer_length;i++)
    {
        ptr_to_buffer[0][i] = (JSAMPLE)jpeg_RGB_Buffer[counter];
        counter++;
    }

    jpeg_write_scanlines(&c_struct, ptr_to_buffer, 1);
}

// Last Step (6) : Clean up
jpeg_finish_compress(&c_struct);

return(ptr_to_buffer[0]);
c_struct.dest->outbuffer=NULL;

jpeg_destroy_compress(&c_struct);

free(ptr_to_buffer[0]);

```

```

*length = (int)c_struct.index;
*ret_quality = (int)n_ImageQuality;

free(temp_RGB_Buffer);

return (JERR)A(returnbuffer);

} // end of method : compute()

// -----
// ##### DECODER
// -----

BYTE * Jpeg_Decode(BYTE *CompressedBuffer, int length){

int i, count=0;
JSAMPLE *temp_buffer;

JSAMPLE *temp_buffer[1]; /* temporary buffer to read in the scanlines. */

int buffer_length;

/* First Step: assigning structure variables */
struct jpeg_decompress_struct d_struct; /* decompression structure */
struct jpeg_error_mgr jerr; /* error handling stuff */

d_struct.err = jpeg_std_error(&jerr);

/* ### error handler ( Will check later) */

/* Next Step: initialize decompression structure variable */
jpeg_create_decompress(&d_struct);

/* ### Making a pointer in the structure to the Compressed Input Data */

/* Next Step: Store file pointer in decompression structure. */
// jpeg_set_src(&d_struct,in_fp);

jpeg_buffer_alloc(&d_struct);

d_struct.dct_buffer=(JSAMPLE *)CompressedBuffer;
d_struct.dct_buffer_length=length;

/* Next Step: Need the info. from the header to decompress the data */
jpeg_read_header(&d_struct,TRUE); /* ### Need to figure out what the "TRUE" does. */

n_ImageWidth = d_struct.image_width;
n_ImageHeight = d_struct.image_height;
n_ImageBpp = d_struct.num_components;

buffer_length = n_ImageWidth*n_ImageBpp; /* width of buffer to take care of RGB values in succession..
.. */

/* buffer is being allocated */
/* Here go... */

temp_buffer = (JSAMPLE *)malloc(buffer_length*sizeof(JSAMPLE));
if (temp_buffer[0]==NULL)
{
fprintf(stderr,"Out of memory for temporary buffer \n");

```



```

        while(1);
    }

    /* Raw buffer */
    raw_buffer=(JSAMPLE *)malloc(n_ImageHeight*buffer_length*sizeof(JSAMPLE ));
    if (raw_buffer==NULL)
    {
        perror(stderr,"Out of memory for raw buffer \n");
        exit(1);
    }

    /* Next Step: signal start decompression */
    jpeg_start_decompress(&d_struct);

    /* Check on dimensions */
    if (!((n_ImageHeight==(int)d_struct.output_height)&&(n_ImageWidth==(int)d_struct.output_width)&&(n_ImageBpp==(int)d_struct.out_color_components)))
    {
        fprintf(stderr,"Image dimensions / bpp do not match\n");
        exit(1);
    }

    /* Next Step: do the actual reading... */
    while ((int)d_struct.output_scanline<n_ImageHeight)
    {
        jpeg_read_scanlines(&d_struct,temp_buffer,1);

        for(i=0; i<buffer_length;i++)
        {
            temp_buffer[counter]=temp_buffer[0][i];
            counter++;
        }

        /* end of while loop.*/
        free(temp_buffer[0]);

        /* Final Step: It finally works!!!
        jpeg_finish_decompress(&d_struct);
        jpeg_decompress_decompress(&d_struct);
        printf("Decompress counter = %d\n",counter);
        // ### temp_buffer free temp_buffer
        return BYTE * raw_buffer;
        */
        /* end of code. */
    }

    /* Method to crop window..... get a central window with 1/7th the original sides. */

    BYTE * cropWindow(BYTE * whole_stream, int * height, int * width, int bpp)
    {
        int row_start, row_end, col_start, col_end;
        int i, diffc, diffb, diffg, diffk=1, diffc, diffb, diffg;
        int ht, wd, c_start, c_end;

        BYTE * crop_diff;

        ht=*height;
        wd=*width;

        row_start=ht/7;
        row_end=ht-ht/7;

        diffc=row_end-row_start+1;

```

```

*height=diff;

c_start=0;
c_end=(row_end-1);

diffc=c_end-c_start+1;

col_start=(c_start-1)*bpp+1;
col_end=c_end*bpp;

diffcb=diff*diffc; // multiply after the calculation. :)

*width=diffcb;

small_buff=(BYTE *)malloc(diffr*diffcb*sizeof(BYTE));
if (small_buff==NULL)
{
    fprintf(stderr,"Memory Alloc error \n");
    exit(1);
}

wdb=wdb*diffcb;

for (i=row_start; i<=row_end; i++)
{
    for (j=col_start; j<=col_end; j++)
    {
        small_buff[s_k-1]=whole_stream[w_k-1];
        s_k++;
        w_k++;
    }
}

return small_buff;

```

/\* Overload encode function to take care of compression ratio \*/

```

BYTE *jpeg_encode(BYTE *jpeg_get_Buffer,int *length, int *ret_quality, int n_Height, int n_Width,
int n_bpp, int n_Ratio, long n_Time, int n_Res, int n_Coding)
{
    BYTE *jpeg_rawBuffer; // for the small test window we're working with .
    BYTE *jpeg_encodeBuffer;
    int len;
    int n_height, n_width, small_img_size;
    int n_store_width;
    float comp_ratio;
    float low_comp_ratio, hi_comp_ratio, mid_comp_ratio;
    int quality_lmt=10, quality_hi_lmt=90, quality;
    int quality_lmt;
    int failed_attempts=0;
    long n_max_time_sec_elapsed;

    time_t start_time, end_time;
    double time_taken;

    BYTE *jpeg_get_Buffer;
    int bpp;
    int res;

```

```

bpp = 3;
res = 100;

jpeg_RGB_Buffer = Resolution_Convertor(jpeg_get_Buffer, &bpp, &res, n_Height, &n_Width);
n_ImageBpp = 3;

n_ImageHeight = n_Height;
n_ImageWidth = n_Width;
n_ImageBpp = n_Bpp;
//add code to assert BPP=1 or 3, and Width and Height >0

n_ImageResolution = n_Res;
n_ImageCodingType = n_Coding;
comp_ratio = n_Ratio; // specifying compression ratio instead of the Quality factor.
max_time = Time; // Maximum # of seconds that will be tolerated for the determination of the
optimum quality factor.

// Code to get I'm taking, say, an n by m window, where n = (1/7)*ImageHeight
// m = (1/7)*ImageWidth*Image_Bpp.

quality_mid_lmt = (quality_hi_lmt + quality_lo_lmt) / 2;

n_height = ImageHeight;
n_width = ImageWidth;

jpeg_Raw_Buffer = ChopWindow(jpeg_RGB_Buffer, &n_height, &n_width, (int)n_ImageBpp);

small_img_size = n_height * n_width * n_ImageBpp;

n_small_img_height = n_ImageHeight;
n_small_img_width = n_ImageWidth;

n_small_img_height = n_height;
n_small_img_width = n_width;

// n_small_img_Buffer = test.Encode(jpeg_Raw_Buffer, &len);

// Code to get the upper and lower bound ratios corresponding to the upper and lower
// bound of the quality factor i.e. 90 and 10 respt.

// Code to get lower bound on the quality factor i.e. 10.

jpeg_Raw_Buffer = Encode(jpeg_SmallRaw_Buffer, &len, ret_quality, (int)n_ImageHeight, (int)n_ImageWidth,
(int)n_ImageBpp, (int)10);
lo_comp_ratio = (float)len / (float)small_img_size;
free(jpeg_Raw_Buffer);

// Code to get upper bound on the quality factor i.e. 90.
jpeg_Raw_Buffer = Encode(jpeg_SmallRaw_Buffer, &len, ret_quality, (int)n_ImageHeight, (int)n_ImageWidth,
(int)n_ImageBpp, (int)90);
hi_comp_ratio = (float)len / (float)small_img_size;
free(jpeg_Raw_Buffer);

// Code to get middle bound on the quality factor i.e. (90-10)/2.
jpeg_Raw_Buffer = Encode(jpeg_SmallRaw_Buffer, &len, ret_quality, (int)n_ImageHeight, (int)n_ImageWidth,
(int)n_ImageBpp, (int)quality_mid_lmt);
mid_comp_ratio = (float)len / (float)small_img_size;
free(jpeg_Raw_Buffer);

// Code for determination of quality factor.

flag = 0;

time(&start_time);

while (1)
{
    if (comp_ratio >= hi_comp_ratio)
    {
        quality = quality_lo_lmt;
    }
}

```



```

BYTE *jpeg_resolution_Convertor(BYTE *jpeg_get_Buffer, int *bpp, int *res, int Height, int *Width)
{
    // must be either 1 or 3.

    int val1, val2;
    int want_res;
    int length, length2, length4, i, diff_len, diff, max, min;

    int val;
    int n_Height, n_Width;
    int diff_len, start_i;

    diff_len = 0;

    BYTE *jpeg_RGB_Buffer;

    have_bpp = *bpp;
    want_res = *res;

    n_Height = Height;
    n_Width = *Width;

    if (want_res < 1)
        want_res = 1;

    *jpeg_RGB_Buffer = 0;

    mod_val = (want_res-1, 4);

    want_res = mod_val.rem+1;

    switch (want_res)
    {
        case ONE_BYTE: {
            if (have_bpp == 3)
                length = n_Height * n_Width;
            else
                length = n_Height * n_Width * have_bpp;

            jpeg_RGB_Buffer = (BYTE *) malloc(sizeof(BYTE) * length);
            if (jpeg_RGB_Buffer == NULL) {
                printf(stderr, "Memory alloc error\n");
                exit(1);
            }

            if (have_bpp == 3)
            {
                for (i=0; i<length; i++)
                {
                    val=0;
                    val+=jpeg_get_Buffer[3*i];
                    val+=jpeg_get_Buffer[(3*i)+1];
                    val+=jpeg_get_Buffer[(3*i)+2];

                    jpeg_RGB_Buffer[i] = (BYTE) (val/3);
                }
            }
            else
            {
                for (i=0; i<length; i++)
                    jpeg_RGB_Buffer[i] = jpeg_get_Buffer[i];
            }

            *Width = n_Width;

            return jpeg_RGB_Buffer;
        }
    }
}

```

```

case TWO_BYTE):{
    len = n_Height*n_Width*have_bpp;
    length2=length/2;

    length2+=(length-(2*length2));

    jpeg_get_Buffer=(BYTE *) malloc(sizeof(BYTE)*len);
    if (jpeg_get_Buffer==NULL){
        fprintf(stderr,"Memory alloc error\n");
        exit(1);
    }

    min=0;
    max=255;
    for (i=0;i<length2;i++)
    {
        val1=(int)jpeg_get_Buffer[i*2];
        val2=(int)jpeg_get_Buffer[(i*2)+1];

        val=(val1*256)+val2;
        if (val>=max)
            max=val;
        if (val<=min)
            min=val;
    }

    // the last guy
    if (length!=(2*length2))
    {
        val=(int)jpeg_get_Buffer[2*length2];

        if (val>=max)
            max=val;
        if (val<=min)
            min=val;

        diff=max-min;

        for (i=length2;i<length2+i++)
        {
            val1=(int)jpeg_get_Buffer[i*2];
            val2=(int)jpeg_get_Buffer[(i*2)+1];

            val=(val1*256)+val2;

            jpeg_get_Buffer[i]=(BYTE)((val-min)*255/diff);
        }

        if (length!=(2*length2))
        {
            val=(int)jpeg_get_Buffer[2*length2];

            jpeg_get_Buffer[length2]=(BYTE)((val-min)*255/diff);
        }
    }

case THREE_BYTE):{
    len = n_Height*n_Width*have_bpp;
    length=
    length/((n_Height*3));
    if ((length*3*n_Height)!=len)
        ++;
}

```

```

        f_len=(wid*3*n_Height)-len;
        n+=wid_diff_len;

        B_Buffer=(BYTE *) malloc(sizeof(BYTE)*(len));
        if (g_RGB_Buffer==NULL){
            printf(stderr,"Memory alloc error\n");
            return(1);
        }

        for (i<length;i++)
            g_RGB_Buffer[i]=jpeg_get_Buffer[i];

        for (i=length;i<len;i++)
            g_RGB_Buffer[i]=0;

        return wid;
    }
}

```

```

case (4) : (FOUR_BYTE) : {
    n_Height*n_Width*have_bpp;
    n=length/4;
    length4;
    length!=(4*length4))
        n=len+3;

    n=(n_Height*3);
    if (d*3*n_Height)!=len)
        ++;

    f_len=(wid*3*n_Height)-len;
    n+=wid_diff_len;

    B_Buffer=(BYTE *) malloc(sizeof(BYTE)*len);
    if (g_RGB_Buffer==NULL){
        printf(stderr,"Memory alloc error\n");
        return(1);
    }

    for (i<length4;i++)
        g_RGB_Buffer[i*3]=jpeg_get_Buffer[4*i];
        g_RGB_Buffer[(i*3)+1]=jpeg_get_Buffer[(4*i)+1];
        g_RGB_Buffer[(i*3)+2]=jpeg_get_Buffer[(4*i)+2];

    n=length-(4*length4);

    for (i<diff_len;i++)
        g_RGB_Buffer[(length4*3)+i]=jpeg_get_Buffer[(length4*4)+i];

    n=(3*length4)+diff_len;

    for (start_i;i<len;i++)
        g_RGB_Buffer[i]=0;

    return wid;
}
}

```

do {
 }

```
        ) // end of the switch statement.  
  
    return (int) _Buffer;  
  
} // end of Method Resolution_Convertor.
```



```

/*****
*
* Copyright (C) 2000, Louisiana State University, School of Medicine
*
* This DICOM object library was developed based on University of California,
* Davis UCSD DICOM Network Transport Libraries, in full compliance
* with the copyright note below. This version however contains conceptual
* deviations from the UCDCM library, as well as important bug and performance
* fixes, and cannot be used/copied/distributed without our permission
*
* Technical Contact: oleg@bit.csc.lsu.edu
*
*****/

```

```

/*****
* PDU Service Classes:
* A-ASSOCIATION-RQ Class.
*
* Base Classes:
* Application Context
* Abstract Syntax
* Transfer Syntax
* Presentation Context
* Maximum Length
* ImplementationClass
* ImplementationVersion
* UserInformation
*
*
*****/

```

```

#ifndef _ARQ_HPP_INCLUDED_
#define _ARQ_HPP_INCLUDED_

class ApplicationContext
{
private:
    BYTE ItemType; // 0x10
    BYTE Reserved1; // 0x00
    UIN Length;
public:
    UIN ApplicationContextName;

    ApplicationContext();
    ApplicationContext(UIN &);
    ApplicationContext(BYTE *);
    ~ApplicationContext();
    void Set(UIN &);
    void Set(BYTE *);
    BOO Write(Buffer &);
    BOO Read(Buffer &);
    BOO ReadDynamic(Buffer &);
    UIN Size();
};

```

```

class AbstractSyntax
{
private:
    BYTE ItemType; // 0x30
    BYTE Reserved1; // 0x00
    UIN Length;
public:
    UIN AbstractSyntaxName;

    AbstractSyntax();
    AbstractSyntax(BYTE *);
    AbstractSyntax(UIN &);
    ~AbstractSyntax();
    void Set(UIN &);
    void Set(BYTE *);
    BOO Write(Buffer &);
    BOO Read(Buffer &);
    BOO ReadDynamic(Buffer &);
    UIN Size();
};

```

```

class TransferSyntax
{
private:
    BYTE ItemType; // 0x40

```

```

    BYTE Reserved1; // 0x00
    UINT Length;
public:
    UINT EndianType; // not really used so far
    UID TransferSyntaxName;

    TransferSyntax();
    TransferSyntax(BYTE *);
    TransferSyntax(UID &);
    ~TransferSyntax();
    void Set(UID &);
    void Set(BYTE *);
    void SetType(UINT T) { EndianType = T; };
    BOC Write(Buffer &);
    BOC Read(Buffer &);
    BOC ReadDynamic(Buffer &);
    UINT Size();
};

```

```

class ImplementationClass
{
private:
    BYTE ItemType; // 0x52
    BYTE Reserved1; // 0x00
    UINT Length;
public:
    UID ImplementationName;

    ImplementationClass();
    ImplementationClass(BYTE *);
    ImplementationClass(UID &);
    ~ImplementationClass();
    void Set(UID &);
    void Set(BYTE *);
    BOC Write(Buffer &);
    BOC Read(Buffer &);
    BOC ReadDynamic(Buffer &);
    UINT Size();
};

```

```

class ImplementationVersion
{
private:
    BYTE ItemType; // 0x55
    BYTE Reserved1; // 0x00
    UINT Length;
public:
    UID Version;

    ImplementationVersion();
    ImplementationVersion(BYTE *);
    ImplementationVersion(UID &);
    ~ImplementationVersion();
    void Set(UID &);
    void Set(BYTE *);
    BOC Write(Buffer &);
    BOC Read(Buffer &);
    BOC ReadDynamic(Buffer &);
    UINT Size();
};

```

```

class SCURoleSelect
{
private:
    BYTE ItemType; // 0x54
    BYTE Reserved1; // 0x00
    UINT Length;
public:
    BYTE SCURole;
    BYTE SCPRole;
    UID SOPuid;

    SCURoleSelect();
    ~SCURoleSelect();
    BOC Write(Buffer &);
    BOC Read(Buffer &);
    BOC ReadDynamic(Buffer &);
    UINT Size();
};

```

```

};

class PresentationContext
{
private:
    BYTE ItemType; // 0x20
    BYTE Reserved1; // 0x00
    BYTE Reserved2; // 0x00
    BYTE Reserved3; // 0x00
    BYTE Reserved4; // 0x00
    UINT Length;

public:
    BYTE PresentationContextID;
    AbstractSyntax AbsSyntax;
    ArrangedTransferSyntax TrnSyntax;

    PresentationContext();
    PresentationContext(AbstractSyntax &, TransferSyntax &);
    ~PresentationContext();
    void SetAbstractSyntax(AbstractSyntax &);
    void AddTransferSyntax(TransferSyntax &);
    BOC Write(Buffer &);
    BOC Read(Buffer &);
    BOC ReadDynamic(Buffer &);
    UINT Size();
};

```

```

class MaximumSubLength
{
private:
    BYTE ItemType; // 0x51
    BYTE Reserved1; // 0x00
    UINT Length; // 0x04
    UINT MaximumLength;

public:
    MaximumSubLength();
    MaximumSubLength(UINT32);
    ~MaximumSubLength();
    Set(UINT32);
    Get();
    BOC Write(Buffer &);
    BOC Read(Buffer &);
    BOC ReadDynamic(Buffer &);
    UINT Size();
};

```

```

class ExtendedNegotiation
{
private:
    BYTE ItemType; // 0x56
    BYTE Reserved1; // 0x00
    BYTE RelationalDB;
    UINT Length;

public:
    UINT SOPuid;

    ExtendedNegotiation();
    ~ExtendedNegotiation();
    BOC Write(Buffer &);
    BOC Read(Buffer &);
    BOC ReadDynamic(Buffer &);
    UINT Size();
};

```

```

class UserInformation
{
private:
    BYTE ItemType; // 0x50
    BYTE Reserved1;
    UINT Length;

public:
    UINT UserInfoBaggage;
    MaximumSubLength MaxSubLength;
    ImpPresentationClass ImpClass;
    ImpPresentationVersion ImpVersion;
    SCPRoleSelect SCPSCURole;
    ExtendedNegotiation ExtNegotiation;
};

```

```

    UserInformation();
    ~UserInformation();
    void SetMax(MaximumSubLength &);
    UINT GetMax();
    BOOL Write(Buffer &);
    BOOL Read(Buffer &);
    BOOL ReadDynamic(Buffer &);
    UINT Size();
};

class AAAssociateRQ
{
private:
    BYTE ItemType; // 0x01
    BYTE Reserved1;
    UINT Length;
    UINT ProtocolVersion; // 0x01
    UINT Reserved2;
public:
    BYTE CalledApTitle[17]; // 16 bytes transferred
    BYTE CallingApTitle[17]; // 16 bytes transferred
    BYTE Reserved3[32];

    ApplicationContext ApplicationContext;
    Array<PresentationContext> PresContexts;
    UserInformation UserInfo;
public:
    AAAssociateRQ();
    AAAssociateRQ(BYTE *, BYTE *);
    virtual ~AAAssociateRQ();
    void SetCalledApTitle(BYTE *);
    void SetCallingApTitle(BYTE *);
    void SetApplicationContext(ApplicationContext &);
    void SetApplicationContext(UINT &);
    void AddPresentationContext(PresentationContext &);
    void ClearPresentationContexts();
    void SetUserInformation(UserInformation &);
    BOOL Write(Buffer &);
    BOOL Read(Buffer &);
    BOOL ReadDynamic(Buffer &);
    UINT Size();
};

#endif

```

```

/*****
*
* Copyright (C) 2000, Louisiana State University, School of Medicine
*
* This DICOM object library was developed based on University of California,
* Davis UCDIC DICOM Network Transport Libraries, in full compliance
* with the copyright note below. This version however contains conceptual
* deviations from the UCDIC library, as well as important bug and performance
* fixes, and cannot be used/copied/distributed without our permission
*
* Technical Contact: oleg@bit.csc.lsu.edu
*
*****/

template <class DATATYPE>
class DataLink
{
public:
    DATATYPE Data;
    DataLink<DATATYPE> *prev, *next;

    DataLink() { prev = NULL; next = NULL; };
};

template <class DATATYPE>
class Array
{
private:
    unsigned int LastAccessNumber;
    unsigned int ArraySize;
    DataLink<DATATYPE> *first;
    DataLink<DATATYPE> *last;
    DataLink<DATATYPE> *LastAccess;

public:
    UINT ClearType;

    void RemoveAll()
    {
        if(ClearType == 1) { while(ArraySize) RemoveAt(0); }
    }

    void RemoveLast()
    {
        if(ArraySize<1) return;
        RemoveAt(ArraySize-1);
    }

    void SetSize(int new_size);
    void Swap(UINT index1, UINT index2);
    void Include(Array<DATATYPE>& a);
    bool Empty() { return (ArraySize<=0); };
    BOOL RemoveAt(unsigned int);
    BOOL ClearArray ()
    {
        first = last = LastAccess = NULL;
        LastAccessNumber = 0;
        ArraySize = 0;
        return ( TRUE );
    }

    inline unsigned int GetSize() { return ArraySize ; };
    inline int GetUpperBound()
    {
        if (ArraySize<=0) return -1;
        else return ArraySize-1;
    }

    DATATYPE& operator[](DATATYPE&);
    DATATYPE& operator[]();
    DATATYPE& operator[](unsigned int);
    inline DATATYPE& operator[] (unsigned int Index)
    {
        return(Get(Index));
    }
};

```

```

virtual ~Array()
{
    RemoveAll();
}

void operator = (Array<DATATYPE> &array)
{
    RemoveAll();
    first = array.first; last = array.last;
    ArraySize = array.ArraySize; ClearType = FALSE;
}

// Constructors
Array()
{
    ArraySize = 0; first = NULL; last = NULL;
    LastAccess = NULL; LastAccessNumber = 0;
    ClearType = 1;
}

Array(T CT)
{
    ArraySize = 0; first = NULL; last = NULL;
    LastAccess = NULL; LastAccessNumber = 0;
    ClearType = CT;
}

Array(DATATYPE & d)
{
    ArraySize = 0; first = NULL; last = NULL;
    LastAccess = NULL; LastAccessNumber = 0;
    ClearType = 1;
    Add(d);
}

private:
void Move(DataLink<DATATYPE> *dmove, DataLink<DATATYPE> *dloc,
          bool insert_before_dloc);
void Swap(DataLink<DATATYPE> *d1, DataLink<DATATYPE> *d2);
inline DataLink<DATATYPE> *
GetDataLink(unsigned int Index);
}; // end of class prototype
/*****
*****
***** Element Manipulation
*****
***** */
template<class DATATYPE>
DATATYPE Array<DATATYPE> :: Add(DATATYPE &Value)
{
    // record current end-of-chain element
    DataLink<DATATYPE> *dl = last;

    // chain new element at tail of chain
    last = new DataLink<DATATYPE>;
    last->Data = Value;

    // set element's backward pointer to point to former
    // end-of-chain element
    last->Back = dl;

    // set former end-of-chain's next pointer to point to new element
    if(dl) dl->next = last;
    else
    {
        // there was previously no "last" element so the one just
        // located must be the first
        first = last;
    }

    ++ArraySize;
    return Value;
}

template<class DATATYPE>
DATATYPE Array<DATATYPE> :: Add()
{
    // record current end-of-chain element
    DataLink<DATATYPE> *dl = last;

    // chain new element at tail of chain
    last = new DataLink<DATATYPE>;
}

```

```

// set element's backward pointer to point to former
// end-of-chain element
last->prev = dl;

// set former end-of-chain's next pointer to point to new element
if(dl->next == 0)
else
{
    // there was previously no "last" element so the one just
    // allocated must be the first
    first = last;
}

++ArraySize;
return last->Data;
}

template <class DATATYPE>
DATATYPE Array<DATATYPE> :: Get(unsigned int Index)
{
    return GetDataLink(Index)->Data; // will throw exception on NULL link
}

template <class DATATYPE>
DataLink<DATATYPE> * Array<DATATYPE> :: GetDataLink(unsigned int Index)
{
    if ( Index >= ArraySize ) return NULL;

    if ( LastAccess ) // Sort of access cache
    {
        int d = (int)LastAccessNumber-Index;
        if (d == 0) // same as before
        {
            return LastAccess;
        }
        else if (d == -1) // next after the most recently accessed
        {
            LastAccess = LastAccess->next;
            LastAccessNumber;
            return LastAccess;
        }
        else if (d == 1) // previous before the most recently accessed
        {
            LastAccess = LastAccess->prev;
            LastAccessNumber;
            return LastAccess;
        }
    }

    // locate requested element by following pointer chain
    // decide which is faster -- scan from head or scan from tail
    DataLink<DATATYPE> *dl;
    unsigned int rIndex = Index;

    if (Index < ArraySize / 2)
    {
        // requested element closer to head -- scan forward
        dl = first;
        while (dl->next != 0)
        {
            ++dl;
            if (--Index > 0) dl = dl->next;
        }
    }
    else
    {
        // requested element closer to tail -- scan backwards
        dl = last;
        Index = (ArraySize - Index);
        while (Index > 0) dl = dl->prev;
    }
    LastAccess = dl;
    LastAccessNumber = rIndex;
    return dl;
}

template <class DATATYPE>
BOOL Array<DATATYPE> :: RemoveAt(unsigned int Index)
{
    DataLink<DATATYPE> *dl = GetDataLink(Index);

```

```

if(!dl) return FALSE;

// Relink chain around element to be deleted
if(dl->prev) dl->prev->next = dl->next;
else first = dl->next;

if(dl->next) dl->next->prev = dl->prev;
else last = dl->prev;

delete dl;
--ArraySize;
LastAccess = NULL;
LastAccessNumber = 0;
return TRUE;
}
/*****
*
* Include elements from Array "a" into this array.
* Note: Array "a" becomes empty after inclusion !
*
*****/
template<class DATATYPE>
void Array<DATATYPE>:: Include(Array<DATATYPE>& a)
{
    if(!ClearType || !a.ClearType) return; // Cannot include linked arrays
    if(a.ArraySize<=0) return; // Nothing to include
    if(ArraySize>0)
    {
        last->next=a.first; a.first->prev = last;
        last = a.last;
    }
    else
    {
        first = a.first; last = a.last;
    }
    ArraySize += a.ArraySize;
    LastAccess = NULL;
    LastAccessNumber = 0;
    a.ClearType = 0;
    a.ClearArray();
}
/*****
*
* Move one linked element before or after the other
*
*****/
template<class DATATYPE>
void Array<DATATYPE>:: Move(DataLink<DATATYPE> *dmove,
DataLink<DATATYPE> *dloc, bool insert_before_dloc)

{
    if(!dmove || !dloc || dmove==dloc) return;

    // Do we need to move anything at all?
    if(insert_before_dloc)
    {
        if(dmove->next==dloc) return; // already there
    }
    else // insert after dloc
    {
        if(dmove->prev==dloc) return; // already there
    }

    // Remove dmove from its present location
    if(dmove->prev) dmove->prev->next = dmove->next;
    else first = dmove->next;

    if(dmove->next) dmove->next->prev = dmove->prev;
    else last = dmove->prev;

    // Insert dmove into its new location
    if(insert_before_dloc)
    {
        dmove->prev = dloc->prev;
        dmove->next = dloc;
        if(dloc->prev) dloc->prev->next = dmove;
        else first = dmove;
        dloc->prev = dmove;
    }
    else // move after dloc
    {

```



```

        dmove->next = dloc->next;
        dmove->prev = dloc;
        if(dmove->next) dloc->next->prev = dmove;
        else last = dmove;
        dloc->next = dmove;
    }
    LastAccessNumber = 0;
    LastAccess = NULL;
}

/*****
 *
 * Swapping two elements
 *
 *****/
template<class DATATYPE>
void Array<DATATYPE>:: Swap(DataLink<DATATYPE> *d1,
                           DataLink<DATATYPE> *d2)
{
    if(!d1 || !d2 || d1==d2) return;
    DataLink<DATATYPE> *dt = d1->next;
    if(dt)
    {
        if(d1==d2) // neighbors
        {
            Move(d1, d2->next, true);
            Move(d2, d1->prev, false);
            // array contains only d1 and d2

            first=d2; last=d1;
            d2->next = d1; d2->prev = NULL;
            d1->prev = d2; d1->next = NULL;
        }
        else // not neighbors
        {
            Move(d1,d2,true); Move(d2,dt,true);
        }
    }
    else
    {
        dt = d2->next; if(!dt) return; // impossible
        if(d1==d2) // neighbors
        {
            Move(d2, d1->next, true);
            Move(d1, d2->prev, false);
            // array contains only d1 and d2

            first=d1; last=d2;
            d1->next = d2; d1->prev = NULL;
            d2->prev = d1; d2->next = NULL;
        }
        else // not neighbors
        {
            Move(d2,d1,true); Move(d1,dt,true);
        }
    }
    LastAccessNumber = 0;
    LastAccess = NULL;
}

template<class DATATYPE>
void Array<DATATYPE>:: Swap(UINT index1, UINT index2)
{
    Swap(GetDataLink(index1), GetDataLink(index2));
}

/*****
 *
 * Setting a certain size
 *
 *****/
template<class DATATYPE>
void Array<DATATYPE>:: SetSize(int new_size)
{
    if(new_size < 0) return;
    if(new_size == 0) { RemoveAll(); return; };
    int n;
    int d = arraySize - new_size;
    if(d > 0) Remove last elements

```

```

    {
        for (n<d; n++) RemoveLast();
        return;
    }
    if(d<0) Add new default elements
    {
        for (n<-d; n++) Add();
        return;
    }
    return; arraySize == new_size
}

```



```

{
private:
    BYTE    uid[65];
    UINT    Length;
public:
    void ClearUID() { ZeroMem(uid, 64); Length = 0; };
    void SetUID(const BYTE *s)
    {
        if(!s) return;
        ZeroMem(uid, 64);
        strcpy((char *) uid, (char *) s);
        Length = strlen((char *) uid);
    };
    void SetUID(const UID &u) { (*this) = u; };
    void SetUID(const char *s) { this->Set((BYTE *) s); };
    void SetLength(UINT L)
    {
        Length = L;
        while (L < 65) uid[L++] = '\0';
    };
    BOOL operator == (const UID &u)
    {
        if(!strcmp((char *) GetBuffer(), (char *) u.GetBuffer()))
            return(TRUE);
        return(FALSE);
    };
    BOOL operator != (const UID &u) { return (!(*this)==u); };
    BOOL operator = (const UID &u)
    {
        ByteCopy(uid, u.GetBuffer(), 64);
        SetLength(u.GetSize());
        return(TRUE);
    };
    BYTE *GetBuffer() { return(&uid[0]); };
    UINT GetSize() { return (Length); };

    UID() { ClearUID(); };
    UID(BYTE *s) { Set(s); };
    UID(char *s) { Set((BYTE *)s); };
};

// DateTime
class DateTime
{
public:
    void SetNumericTime(double t);
    double GetNumericTime();
    void SetNumericDate(int dt);
    int GetNumericDate();
    static const BYTE DateFormat;
    static const BYTE TimeFormat;
    static const BYTE DateTimeFormat;
    static const BYTE UnknownFormat;

    void SetCurrentDateTime();
    void SetCurrentTime();
    void SetCurrentDate();
    inline void ClearDate() { m_Year=-1; m_Month=0; m_Day=0; };
    inline void ClearTime() { m_Hour=-1; m_Minute=0; m_Second=0.0; };
    inline void ClearDateTime() { ClearDate(); ClearTime(); };
    inline void EmptyDate() { return (m_Year<0); };
    inline void EmptyTime() { return (m_Hour<0); };
    inline void EmptyDateTime() { return EmptyDate()||EmptyTime(); };
    bool SerializeDateTime(FILE* fp, bool is_loading);
    bool SetTime(char* str);
    bool SetDate(char *str);
    bool SetDateTime(int y, int mo=0, int d=0, int h=0,
                     int mi=0, double s=0);
    bool SetTime(int h, int m=0, double s=0);
    bool SetDate(int y, int m=0, int d=0);
    bool SetSecond(double s);
    bool SetMinute(int m);
    bool SetHour(int h);
    bool SetDay(int d);
    bool SetMonth(int m);
    bool SetYear(int y);
    bool FormatDateTime(char *str, int max_len, bool dicom_format);
    bool FormatDate(char *str, int max_len, bool dicom_format);
    bool FormatTime(char *str, int max_len, bool dicom_format);
};

```

```

int      SetDateTime(char* str, BYTE format=UnknownFormat);
int      GetYear()    { return m_Year; };
int      GetMonth()   { return m_Month; };
int      GetDay()     { return m_Day; };
int      GetHour()    { return m_Hour; };
int      GetMinute()  { return m_Minute; };
double   GetSecond()  { return m_Second; };
struct tm GetTM();
DateTime GetLower();
DateTime GetUpper();

bool      operator == (DateTime& d);
bool      operator != (DateTime& d);
bool      operator > (DateTime& d);
bool      operator >= (DateTime& d);
bool      operator < (DateTime& d);
bool      operator <= (DateTime& d);

DateTime& operator = (DateTime& d);
virtual ~DateTime();

private:
int      m_Year, m_Month, m_Day, m_Hour, m_Minute;
double   m_Second;

bool      Format(char *dest, int max_len, const char *format);
};

// DateTimeSegment
class DateTimeSegment
{
public:
void      SetNumericTime(double t);
void      SetNumericDate(int d);
void      Normalize();
inline void SetStart(DateTime& d)    { m_Start=d; };
inline void SetEnd(DateTime& d)      { m_End=d; };
inline void SetDateTimeSegment(DateTime& start, DateTime& end)
        { m_Start=start; m_End=end; };
inline void ClearDateTimeSegment()
        { m_Start.ClearDateTime(); m_End.ClearDateTime(); };
bool      SetDateTimeSegment(char* str, BYTE format=DateTime::UnknownFormat);
bool      Intersects(DateTimeSegment& d);
bool      ContainsDateTime(DateTime& dt);
bool      SerializeDateTimeSegment(FILE *fp, bool is_loading);
bool      FormatDateTime(char *str, int max_len, bool dicom_format);
bool      FormatTime(char *str, int max_len, bool dicom_format);
bool      FormatDate(char *str, int max_len, bool dicom_format);
bool      EmptyDateTimeSegment()
        { return (m_Start.EmptyDateTime() && m_End.EmptyDateTime()); };
static char* FormatStaticDateTimeString(char* strdest, int max_dest_len,
        int dest_format, char* strsource, BYTE source_format);
DateTime  GetStart()    { return m_Start; }
DateTime  GetEnd()      { return m_End; }
DateTimeSegment& Expand();

DateTimeSegment();
DateTimeSegment(DateTimeSegment& dts);
virtual ~DateTimeSegment();

private:
DateTime  m_Start, m_End;
};

// ApplicationEntity
class ApplicationEntity
{
public:
bool      a_reserved;
bool      a_useMoveAsGet;
char      a_Title[20], a_partnerTitle[20];
char      a_Location[32];
char      a_Comments[64];
int       a_Port, a_PortServer;
int       a_Timeout;
BYTE      a_IP1, a_IP2, a_IP3, a_IP4;

```

```

bool    SetComments(char* s);
bool    SetLocation(char* s);
bool    SetPartnerTitle(char* s);
bool    SetTitle(char* s);
bool    SetApplicationEntity(char* title,
                               BYTE ip1=127, BYTE ip2=0, BYTE ip3=0, BYTE ip4=1,
                               int port=104, int port_ser=104, int timeout=300,
                               char* location="Unspecified location",
                               char* comments="No comments",
                               bool useMoveAsGet=false);

bool    SerializeAE(FILE* fp, bool is_loading);
bool    operator == (ApplicationEntity& a)
{
    return (strcmp(ae_Title, a.ae_Title)==0);
};
ApplicationEntity();
ApplicationEntity(ApplicationEntity& a);
ApplicationEntity(char* title,
                   BYTE ip1=127, BYTE ip2=0, BYTE ip3=0, BYTE ip4=1,
                   int port=104, int port_ser=104, int timeout=300,
                   char* location="Unspecified location",
                   char* comments="No comments",
                   bool useMoveAsGet=false);

virtual ~ApplicationEntity();

char*    GetPortString();
char*    GetPortServerString();
char*    GetIPString();

private:
    char    ae_IP_str[20], ae_Port_str[8], ae_PortServer_str[8];
};

// ApplicationEntityList
class ApplicationEntityList : public Array<ApplicationEntity>
{
public:
    void    SetServedStatus(int port, bool status);
    bool    GetAELocation(UINT aeindex, char* loc);
    bool    SetLocalAE(BYTE ip1, BYTE ip2, BYTE ip3, BYTE ip4,
                       char* title=NULL);
    bool    IdentifyAE(char* title, ApplicationEntity& aeFound);
    bool    SerializeAEList(bool is_loading, char* filename=NULL);
    bool    SetCurrentIndex(int n);
    int     IdentifyAEIndex(char* title);
    UINT    GetCurrentIndex()    {    return m_CurrentIndex;    };
    UINT    GetLocalIndex()      {    return m_LocalIndex;      };
    ApplicationEntity&
    GetLocalAE()                {    return Get(m_LocalIndex);    };
    ApplicationEntity&
    GetCurrentAE()
    {
        Get(m_CurrentIndex).SetPartnerTitle(GetLocalAE().ae_Title);
        return Get(m_CurrentIndex);
    };
    ApplicationEntityList();
    virtual ~ApplicationEntityList();

private:
    char    m_SerFile[MAX_PATH];
    UINT    m_CurrentIndex;
    const UINT m_LocalIndex;    // always 0

    void    LoadDefaults();
    bool    SetLocalAE(UINT n);
    bool    CreateLocalAE(BYTE ip1, BYTE ip2, BYTE ip3, BYTE ip4,
                          char* title=NULL);
};

#endif // !defined(_BASICTYPES_H_INCLUDED_)

```

```

/*****
 *
 * Copyright (C) 2000, Louisiana State University, School of Medicine
 *
 * This DICOM object library was developed based on University of California,
 * Davis UCDM DICOM Network Transport Libraries, in full compliance
 * with the copyright note below. This version however contains conceptual
 * deviations from the UCDM library, as well as important bug and performance
 * fixes, and cannot be used/copied/distributed without our permission
 *
 * Technical Contact: oleg@bit.csc.lsu.edu
 *
 *****/

```

```

class BufferSpace
{
public:
    BOOL        isTemp;
    BYTE        *Data;
    INT         BufferSize;
    UINT        Index;

    BufferSpace(UINT);
    BufferSpace();
    ~BufferSpace();
};

class Buffer
{
protected:
    UINT        BreakSize;
    UINT        InEndian;
    UINT        OutEndian;
    INT         InSize;
    INT         OutSize;
    Array<BufferSpace*> Incoming;
    Array<BufferSpace*> Outgoing;

    BOOL        ReadBlock();

public:
    BOOL        SetBreakSize(UINT);
    BOOL        SetIncomingEndian(UINT);
    BOOL        SetOutgoingEndian(UINT);
    BOOL        Flush();
    BOOL        Flush(UINT Bytes);
    BOOL        Kill(UINT);
    BOOL        Read(BYTE *, UINT);
    BOOL        Write(BYTE *, UINT);
    BOOL        Fill(UINT);
    inline UINT GetIncomingEndian() { return ( InEndian ); };
    inline UINT GetOutgoingEndian() { return ( OutEndian ); };
    inline UINT GetSize() { return ( InSize ); };

    Buffer & operator >> (BYTE &);
    Buffer & operator >> (UINT16 &);
    Buffer & operator >> (UINT32 &);
    inline Buffer & operator >> (char &x)
    { return ( (*this)>>(BYTE &) x); };
    inline Buffer & operator >> (INT16 &x)
    { return ( (*this)>>(UINT16 &) x); };
    inline Buffer & operator >> (INT32 &x)
    { return ( (*this)>>(UINT32 &) x); };

    Buffer & operator << (BYTE &);
    Buffer & operator << (UINT16 &);
    Buffer & operator << (UINT32 &);
    inline Buffer & operator << (char &x)
    { return ( (*this)<<(BYTE &) x); };
    inline Buffer & operator << (INT16 &x)
    { return ( (*this)<<(UINT16 &) x); };
    inline Buffer & operator << (INT32 &x)
    { return ( (*this)<<(UINT32 &) x); };

    virtual INT ReadBinary(BYTE *, UINT) = 0;
    virtual BOOL SendBinary(BYTE *, UINT) = 0;

    Buffer();
    virtual ~Buffer();
};

```

[illegible]



```

/*****
*
* Copyright (C) 2000, Louisiana State University, School of Medicine
*
* This DICOM object library was developed based on University of California,
* Davis UCDC DICOM Network Transport Libraries, in full compliance
* with the copyright note below. This version however contains conceptual
* deviations from the UCDC library, as well as important bug and performance
* fixes, and cannot be used/copied/distributed without our permission
*
* Technical contact: oleg@bit.csc.lsu.edu
*
*****/

```

```

#if !defined(_CCTYPES_H_INCLUDED_)
#define _CCTYPES_H_INCLUDED_

```

```

/* CC Types */

```

```

#ifdef SOLARIS
# define SVSEM_V
#endif

```

```

#ifdef WIN32
typedef unsigned int BOOL;
#endif
typedef unsigned int UINT;
typedef unsigned short UINT16;
#ifdef _BASETSD_H_ // MSVC 6.0 define: typedef unsigned int UINT32 [gz]
typedef unsigned long UINT32;
#endif
typedef unsigned char UINT8;
#ifdef WIN32
typedef unsigned char BYTE;
#endif
typedef signed char INT8;
typedef signed short INT16;
#ifdef _BASETSD_H_ // MSVC 6.0 define: typedef int INT32 [gz]
typedef signed long INT32;
#endif
#ifdef WIN32
typedef signed int INT;
#endif

```

```

#ifdef TRUE
#define TRUE ((UINT) 1)
#endif
#ifdef FALSE
#define FALSE ((UINT) 0)
#endif

```

```

#ifdef LITTLE_ENDIAN
#undef LITTLE_ENDIAN
#endif
#ifdef BIG_ENDIAN
#undef BIG_ENDIAN
#endif

```

```

# define LITTLE_ENDIAN 1
# define BIG_ENDIAN 2
#ifdef NATIVE_ENDIAN
# define NATIVE_ENDIAN LITTLE_ENDIAN
#endif

```

```

#endif

```

```

////////////////////////////////////
// database.h interface for the DICOMRecord class.
//
////////////////////////////////////

#if !defined( DICOM_DATABASE_H_INCLUDED_)
#define _DICOM_DATABASE_H_INCLUDED_

#include "dicom.hpp"
#include "cctypes.h" // Added by ClassView

class DICOMRecord
{
public:
    const static BYTE LevelInvalid;
    const static BYTE LevelPatient;
    const static BYTE LevelStudy;
    const static BYTE LevelSeries;
    const static BYTE LevelImage;

    void SetRoot(BYTE root, DICOMRecord* pDR=NULL);
    void SetRecord(char *pID, char *pName, int pBDate,
        double pBTime, char* stIUnstID,
        char* stID, char* aNum, char* stImNum,
        int stDate, double stTime,
        char* serInstUID, char* mod, char* serNum,
        char* SOPUID, char* imNum, char* fName);
    void SetRecord(char *pID, char *pName, DateTimeSegment* pBDate,
        DateTimeSegment* pBTime, char* stIUnstID,
        char* stID, char* aNum, char* stImNum,
        DateTimeSegment* stDate, DateTimeSegment* stTime,
        char* serInstUID, char* mod, char* serNum,
        char* SOPUID, char* imNum, char* fName);

    void SetRetrieveAEs(char* aelist);
    void ClearAtCurrentLevel();
    void ClearDICOMRecord(bool remove_file=false);
    void PrintRootModality();
    void LoadDateTime();
    void WriteIntoDICOMObject(DICOMObject& dob, DICOMObject* dob_mask=NULL);
    bool RetrievePrimaryKeys();
    bool WriteIntoDICOMQR(DICOMObject& dob, bool cfind);
    bool WriteIntoTextFile(char* filename);
    bool ChangeLevel(bool step_down, BYTE qr_root);
    bool RetrieveAE(char* aetitle);
    bool RemoveRetrieveAE(char* aetitle);
    bool FormatStudyTime(char *str, int max_len, bool dicom_format);
    bool FormatStudyDate(char *str, int max_len, bool dicom_format);
    bool FormatPatientBirthTime(char *str, int max_len, bool dicom_format);
    bool FormatPatientBirthDate(char *str, int max_len, bool dicom_format);
    bool SerializeDICOMRecord(FILE* fp, bool is_loading);
    bool SetRecordOnLevel(DICOMObject& dob);
    bool SetRecord(DICOMObject& DOB, char* filename);
    bool SetRecord(DICOMRecord& d);
    bool SetRecord(char* filename);
    bool MatchDICOMString(char* exact, char* mask,
        bool case_sensitive,
        bool first_level=false);

    bool operator==(DICOMRecord& dr);

    BYTE FindQLLevel();
    BYTE GetQLLevel() { return m_QLLevel; };
    char* GetFileName() { return m_FileName; };
    char* GetRetrieveAEs() { return m_RetrieveAEs; };
    char* GetPatientID() { return m_PatientID; };
    char* GetPatientName() { return m_PatientName; };
    char* GetStudyInstUID() { return m_StudyInstUID; };
    char* GetStudyID() { return m_StudyID; };
    char* GetAccessionNumber() { return m_AccessionNumber; };
    char* GetStudyImagesNum() { return m_StudyImagesNum; };
    char* GetSeriesInstUID() { return m_SeriesInstUID; };
    char* GetModality() { return m_Modality; };
    char* GetSeriesNum() { return m_SeriesNum; };
    char* GetSOPInstUID() { return m_SOPInstUID; };
    char* GetImageNum() { return m_ImageNum; };
    char* GetRetrieveAE(char* aetitle, UINT aetitle_len, UINT n=1);
    int CompareAtLevel(DICOMRecord& dr, BYTE lev=LevelImage);
    UINT GetRetrieveAEsCount();
    DateTimeSegment GetPBirthTime() { return m_PBirthTime; };

```

```

DateTimeS...ent GetPBirthDate() { return m_PBirthDate; };
DateTimeS...ent GetStudyTime() { return m_StudyTime; };
DateTimeS...ent GetStudyDate() { return m_StudyDate; };

DICOMRecord();
DICOMRecord(DICOMRecord& d);
virtual ~DICOMRecord();

private:
const static BYTE InsertUnique;
const static BYTE InsertNonEmpty;
const static BYTE InsertAny;

BYTE m_Level;
char m_Filename[MAX_PATH+1];
char m_RetrieveAEs[65];
char m_PatientID[65];
char m_PatientName[65];
char m_StudyInstUID[65];
char m_StudyID[17];
char m_SessionNumber[17];
char m_StudyImagesNum[13];
char m_SeriesInstUID[65];
char m_Modality[3];
char m_SeriesNum[13];
char m_PInstUID[65];
char m_PageNum[13];
DateTimeS...ent
m_BirthTime, m_PBirthDate, m_StudyTime, m_StudyDate;

void m_SetStringForQLevel(DICOMObject& dob);
bool m_SetString(DICOMObject& dob, UINT16 group, UINT16 element,
char *str, BOOL alloc, BYTE how=InsertAny);

///////////////////////////////////////////////////
// DICOMDataBase class.
///////////////////////////////////////////////////
class DICOMDataBase
{
public:
const static BYTE RetrieveRelational;
const static BYTE RetrieveHierarchical;
const static BYTE MatchRelational;
const static BYTE MatchHierarchical;
const static BYTE RecordFound;
const static BYTE FindMore;
const static BYTE RecordsEnd;
Applicat...EntityList db_AEList;

void EnableDisplayNew(bool b) { db_DisplayRecordsFlag=b; };
void DisplayRecords(bool from_local);
void* StartSearch(DICOMObject& dob, const BYTE how,
char* filename=NULL);
void* StartSearch(char* filename, const BYTE how);
virtual void StopSearch(void* pDR);
virtual void DisplayRecords(Array<DICOMRecord> &a, bool from_local);
bool DBAdd(char* filename, bool copy_into_db_directory=false);
bool AttachDirectory(char *directory,
bool include_subdirectories=true);
bool ImportDirectory(char* directory,
bool include_subdirectories=true);
bool UnImportDirectory(char *directory,
bool include_subdirectories=true);
bool UnAttachDirectory(char *directory,
bool include_subdirectories=true);
virtual void GetFromLocal(DICOMDataObject& ddo_mask);
virtual void SetRecordCount(int c);
virtual void DBAdd(DICOMRecord& dr);
virtual void DBAdd(DICOMObject* dob, PDU_Service* pdu);
virtual void InitializeDataBase(char* directory,
void (*disp)(Array<DICOMRecord>&, bool));
virtual void RetrieveNext(void* pDR, DICOMObject& dob_found);
virtual void MatchNext(void* pDR, DICOMObject &dob_found,
DICOMObject* dob_mask=NULL);
char* GetMostRecentFile()
{ return db_MostRecentRecord.GetFileName(); };
int DBRemove(char* filename, bool use_filename=false);

```

```

virtual      DBRemove(DICOMRecord& dr_mask);
virtual      GetRecordCount();

DICOMData...();
virtual      OMDatabase();

protected:
    char      db_Directory[MAX_PATH+1];
    DICOMRec... db_MostRecentRecord;

virtual      StartSearch(DICOMRecord& dr_mask, const BYTE how);
virtual      DBAddDirectoryContents(char* directory, bool copy_files,
                                     bool include_subdirectories=true);
virtual      DBRemoveDirectoryContents(char* directory,
                                     bool use_filenames, bool include_subdirectories=true);

private:
    void      ..._DisplayRecords)(Array<DICOMRecord> &a, bool from_local);
    bool      ...splayRecordsFlag;
    char      ...ecFile[MAX_PATH+1];
    char      ...ffFile[MAX_PATH+1];
    int       ...Records;
    FILE*     ...RecFile;

    bool      ...alizeDBHeader(bool is_loading);
};

#endif // !defined(_DICOM_DATABASE_H_INCLUDED_)

```

```
#include "di    .hpp"
#include "der    ."
```

```

////////////////////////////////////
// DICOMView interface for the DICOMView class.
//
////////////////////////////////////

#ifndef DICOMVIEW_H_INCLUDED_
#define DICOMVIEW_H_INCLUDED_

#include "dicomview.hpp"

// #if _MSC_VER > 1000
// #pragma omp
// #endif // _MSC_VER > 1000

class DICOMView
{
public:
    virtual void Load(const char* pText)=0;
    void Load(VR *vr);
    virtual void Load(DICOMObject& DO);
    void ReportTime();
    void AttachRTC(RTC* rtc);
    virtual void LoadFile(char* filename);
    virtual bool IgnoreVR(VR* vr) { return false; }
    DICOMView();
    virtual DICOMView();

protected:
    char m_SequenceMark;
    int m_SequenceLevel;
    int m_numElements;
    RTC* m_AttachedRTC;

    void MarkSequence(bool start);
};

////////////////////////////////////
// Interface for the DICOMViewLog class.
//
////////////////////////////////////

class DICOMViewLog : public DICOMView
{
public:
    void Load(const char* pText );
    void Load(DICOMObject& DO) { DICOMView::Load(DO); };
    bool LoadDVL(char* filename, RTC* rtc=NULL);
    DICOMViewLog();
    virtual DICOMViewLog();

protected:
    char* filename;
    FILE* logFile;

    bool IgnoreVR(VR* vr);
    bool RefreshText(char* tbuffer, long max_length);
};

#endif // !defined(DICOMVIEW_H_INCLUDED_)

```



```
// dqr.h: interface for the DICOM query/retrieve class.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "sequenceclass.h"
#include "comtypes.h" // Added by ClassView
#if !defined(DQR_H_INCLUDED_)
#define DQR_H_INCLUDED_

class DQRTask
{
public:
    BYTE m_Type;
    const static BYTE TaskInvalid, TaskGet, TaskMove;
    UINT m_nAE, m_nAEDest;
    DICOMRecord& m_DRdata;
    DateTimeSegment m_ExecTimeSegment;

    void ScheduleTask(DateTime& start, DateTime& end);
    void CountExecution();
    void GrantExecution() { if(m_nExec<1) m_nExec++; };
    void SetExec(UINT e) { m_nExec = min(e,5); };
    bool CanExecuteLater();
    bool CanExecuteNow();
    bool CanExecute();
    bool SerializeTask(FILE *fp, bool is_loading);
    bool SetTask(BYTE type, UINT nAE, DICOMRecord& drdata,
        UINT nAEDest=0, int nExec=1);
    bool SetTask(BYTE type, UINT nAE, DICOMDataObject& ddo,
        UINT nAEDest=0, int nExec=1);
    bool FormatScheduleString(char *str, int max_len);
    int GetExec() { return m_nExec; };
    UINT GetID() { return m_ID; };
    DQRTask() {}
    virtual ~DQRTask();
private:
    int m_nExec;
    UINT m_ID;
    static int m_IDcount;
    void SetUniqueID();
};

// =====
class DQR
{
public:
    void RemoveTaskID(UINT taskID);
    void AddTask(DQRTask& t);
    void Cancel() { m_CancelEnabled=true; };
    void ClearFound() { m_ADOB.RemoveAll(); };
    void SetPriority(BYTE priority);
    void SetRoot(BYTE root);
    void SetLastMessageID(UINT16 id) { m_LastMessageID = id; m_CancelEnabled=false; };
    bool FindAELocation(UINT aeindex, char* loc);
    bool ExecuteNextTask();
    bool OnRootLevel();
    bool OnBottomLevel();
    bool Go();
    bool Stop();
    bool AddRoot(DICOMRecord& dr);
    bool AddNextLevel(UINT dob_index);
    bool AddPreviousLevel();
    bool Add(UINT dob_index, bool queue);
    bool Remove(UINT dob_index, char* ae_dest_title, bool queue);
    bool ExecuteTask(DQRTask& t);
    bool AddTask(UINT tindex, DQRTask& t);
    bool CreateQR(DICOMViewLog* pLog, DICOMDatabase* pDB);
    bool FoundRecord(DICOMRecord& d, UINT index);
    bool GetCurrentAEIndex(UINT ind);
    bool CallBack(DQRCallBack(int (*func)(void* a, UINT param1=0, UINT param2=0),
        void* arg=NULL)) { return m_ControlCallback.SetCallBack(func,arg); };
    bool InitializeQR(FILE* fp, bool is_loading);
    char* GetCurrentAETitle();
    BYTE Priority() { return m_Priority; };
    BYTE Level() { return m_Record.GetQLevel(); };
    int FoundCount() { return m_ADOB.GetSize(); };
};
```



```

int         ~CallbackFilter(UINT stepnum=0, UINT id=0);
int         ~DeleteFromLocalDB(DICOMRecord& dr);
UINT16      LastMessageID() { return m_LastMessageID; };
UINT        CurrentAEIndex() { return m_CurrentAEIndex; };
UINT        TasksSize();
DICOMRecord *GetRecord() { return m_Record; };
Application *EntityList*
AEListPtr();
DQRTask     ~GetTaskPtrFromID(UINT taskID);
DQR();
virtual     ~DQR();

private:
bool         m_CancelEnabled;
bool         m_TaskCanUpdate;
BYTE         m_Root, m_Priority;
UINT16       m_LastMessageID;
UINT         m_CurrentAEIndex, m_CurrentTaskIndex;
DICOMRecord *m_Record, m_RecordRoot;
DICOMDataset m_DOB;
Array<Dataset> m_ADOB;
Array<DQRTask> m_Tasks;
DICOMViewer *m_pLog;
DICOMDataset *m_pDBase;
CallbackObject m_ControlCallback;

void         LockTaskThread();
void         UnLockTaskThread();
};

#endif // !defined(DQR_H_INCLUDED_)

```

```

/*****
*
* Copyright (C) 2000, Louisiana State University, School of Medicine
*
* This DICOM object library was developed from the University of California,
* Davis UCDMC DICOM Network Transport Libraries, in full compliance
* with the free access licence and copyright note below. The DICOM object
* library, however contains conceptual deviations from the UCDMC library,
* as well as important bug and performance fixes, and CANNOT be
* used/copied/distributed/modified without our permission.
*
* Technical Contact: oleg@bit.csc.lsu.edu
*
*****/
/*****
*
* Sorted / Indexed, Fast-Access Array.
*
*****/

```

```

template <class KEYTYPE, class DATATYPE>
class FixedArray
{
public:
    UINT    ArraySize;
    UINT    Top;
    KEYTYPE *KeyTable;
    DATATYPE *DataTable;

    FixedArray (UINT, BOOL);
    ~FixedArray ();
    BOOL      Sort();
    DATATYPE & Add(KEYTYPE &, DATATYPE &);
    DATATYPE & Add(DATATYPE &);
    INT       IndexOf(KEYTYPE &);
    DATATYPE & Get(INT Index);
    BOOL      RemoveAt(INT);
    DATATYPE & operator [] (INT Index)
    { return(Get(Index)); };
    UINT      GetSize()
    { return ( Top ); };
    UINT      GetAllocationSize()
    { return ( ArraySize ); };
};

template <class KEYTYPE, class DATATYPE>
class FixedArrayElement
{
public:
    KEYTYPE Key;
    DATATYPE Data;
    UINT operator > (FixedArrayElement &FAE)
    { return ( Key > FAE.Key ); };
    UINT operator < (FixedArrayElement &FAE)
    { return ( Key < FAE.Key ); };
    UINT operator == (FixedArrayElement &FAE)
    { return ( Key == FAE.Key ); };
};

template <class KEYTYPE, class DATATYPE>
FixedArray<KEYTYPE, DATATYPE> :: FixedArray (
    UINT aSize,
    BOOL useKeys)
{
    ArraySize = aSize;
    Top = 0;
    if ( useKeys )    KeyTable = new KEYTYPE [ aSize ];
    else              KeyTable = NULL;
    DataTable = new DATATYPE [ aSize ];
}

template <class KEYTYPE, class DATATYPE>
FixedArray<KEYTYPE, DATATYPE> :: ~FixedArray ()
{
    if ( KeyTable )
        delete KeyTable;
    if ( DataTable )
        delete DataTable;
}

```

```

template < class KEYTYPE, class DATATYPE>
BOOL FixArray<KEYTYPE, DATATYPE> :: Sort()
{
    UINT Index;
    FixedArray<Element < KEYTYPE, DATATYPE > FAE;
    PQQueue<FixedArrayElement<KEYTYPE, DATATYPE> > PQFAE;

    if ( ! KeyTable )
        return ( FALSE ); // Not a sortable array

    Index = 0;
    while ( Index < Top )
    {
        FAE.Key = KeyTable [ Index ];
        FAE.Data = DataTable [ Index ];
        PQFAE.Push(FAE);
        ++Index;
    }
    Index = 0;
    while ( Index < Top )
    {
        FAE = PQFAE.Pop ();
        KeyTable [ Index ] = FAE.Key;
        DataTable [ Index ] = FAE.Data;
        ++Index;
    }
    return ( TRUE );
}

```

```

template < class KEYTYPE, class DATATYPE>
DATATYPE FixArray<KEYTYPE, DATATYPE> :: Add(
    KEYTYPE &Key,
    DATATYPE &Data)
{
    if (Top == ArraySize)
    {
        if (KeyTable)
            KeyTable [ Top ] = Key;
        if (DataTable)
            DataTable [ Top ] = Data;
        ++Top;
    }
    return ( DataTable [ Top-1 ] );
}

```

```

template < class KEYTYPE, class DATATYPE>
DATATYPE FixArray<KEYTYPE, DATATYPE> :: Add(
    DATATYPE &Data)
{
    if (Top == ArraySize)
    {
        if (DataTable)
            DataTable [ Top ] = Data;
        ++Top;
    }
    return ( DataTable [ Top-1 ] );
}

```

```

template < class KEYTYPE, class DATATYPE>
INT FixArray<KEYTYPE, DATATYPE> :: IndexOf(
    KEYTYPE &Key)
{
    INT Index = (Top / 2);
    INT Shift = (Top / 2 - 1);

    if ( ! KeyTable )
        return ( -1 );

    if ( ! DataTable )
        return ( -1 );

    while ( KeyTable [ Index ] != Key )
    {
        if ( KeyTable [ Index ] > Key )
            Index -= Shift;
        else
            Index += Shift;
    }
    if ( Index >= (INT)Top )

```

```

        Index = Top - 1; break; });
    if ( Index <= 0 )
        Index = 0; break; });
    if ( Shift == 0 )
        break; }
    Shift = Shift / 2;
}
if ( KeyTable [ Index ] == Key )
    return ( Index );

if ( KeyTable [ Index ] < Key )
{
    while ( Index < (INT)Top )
    {
        if ( KeyTable [ Index ] == Key )
            return ( Index );
        if ( KeyTable [ Index ] > Key )
            return ( -1 );
        Index;
    }
    return ( -1 );
}
else
{
    while ( Index >= 0 )
    {
        if ( KeyTable [ Index ] == Key )
            return ( Index );
        if ( KeyTable [ Index ] < Key )
            return ( -1 );
        Index;
    }
    return ( -1 );
}
return ( -1 );
}

template< class KEYTYPE, class DATATYPE>
BOOL FindArray<KEYTYPE, DATATYPE> :: RemoveAt(
    INT Index)
{
    if ( Index >= Top )
        return ( FALSE );

    if ( DataTable )
    {
        if ( Index + 1 != Top )
            memcpy( (void*)&DataTable [ Index ],
                (void*)&DataTable [ Index + 1 ],
                sizeof ( DATATYPE ) * Top - Index - 1);
    }
    if ( KeyTable )
    {
        if ( Index + 1 != Top )
            memcpy( (void*)&KeyTable [ Index ],
                (void*)&KeyTable [ Index + 1 ],
                sizeof ( KEYTYPE ) * Top - Index - 1);
    }
    --Top;
    return ( TRUE );
}

```

```

/*****
 *
 * Copyright (C) 2000, Louisiana State University, School of Medicine
 *
 * This DICOM object library was developed from the University of California,
 * Davis UCDMC DICOM Network Transport Libraries, in full compliance
 * with the free access licence and copyright note below. The DICOM object
 * library however contains conceptual deviations from the UCDMC library,
 * as well as important bug and performance fixes, and CANNOT be
 * used/modified/distributed/modified without our permission.
 *
 * Technical Contact: oleg@bit.csc.lsu.edu
 *
 *****/

```

```

/*****
 *
 * PQueue Class (binary tree - based)
 *
 * ANSI (ANSI C++ Compatible / Templates Required)
 *
 * usage:
 *
 * #include "pqueue.h"
 *
 * PQueue<DataType> VarName;
 *
 * notes:
 *
 * Any Class used as the datatype must support the operators < > =
 *****/

```

```

template <class DATATYPE> class PQueue : public Array<DATATYPE>
{
    DATATYPE dt;
public:
    DATATYPE & Push(DATATYPE &);
    DATATYPE & Pop();
}

```

```

template <class DATATYPE> class PQueueOfPtr : public Array<DATATYPE>
{
    DATATYPE dt;
public:
    DATATYPE & Push(DATATYPE &);
    DATATYPE & Pop();
}

```

```

/*****
 *
 * Template PQueue Class implementation
 *
 *****/

```

```

template <class DATATYPE>
DATATYPE PQueue<DATATYPE> :: Push(DATATYPE &Value)
{
    unsigned int Index;
    //unsigned int Base;
    DATATYPE tdt;

    Array<DATATYPE> :: Add(Value);

    Index = Array<DATATYPE> :: GetSize();
    --Index;
    while( Index )
    {
        if( Array<DATATYPE> :: Get(Index) < Array<DATATYPE> :: Get((Index-1)>>1) )
        {
            dt = Array<DATATYPE> :: Get(Index);
            Array<DATATYPE> :: Get(Index) = Array<DATATYPE> :: Get((Index-1)>>1);
            Array<DATATYPE> :: Get((Index-1)>>1) = tdt;
            Index = (Index-1) >> 1;
        }
        else
            break;
    }
}

```

```

    return Value );
}

template < class DATATYPE>
DATATYPE PQueue<DATATYPE> :: Pop()
{
    DATATYPE tdt;
    ///DATATYPE tdt2;
    ///unsigned int Index;
    unsigned int sorted;
    unsigned int Pick;
    unsigned int Child1;
    unsigned int Child2;
    unsigned int Hole;

    if(Array<DATATYPE>::GetSize()
        return ( dt ); // error

    dt = Array<DATATYPE> :: Get ( 0 );

    if(Array<DATATYPE> :: GetSize() == 1)
    {
        Array<DATATYPE> :: RemoveAt ( 0 );
        return ( dt );
    }
    tdt = Array<DATATYPE> :: Get ( Array<DATATYPE> :: GetSize() - 1);
    Array<DATATYPE> :: RemoveAt ( Array<DATATYPE> :: GetSize() - 1);
    Hole = 1;
    sorted = 0;
    while(!sorted)
    {
        Child1 = 2 * Hole + 1;
        Child2 = 2 * Hole + 2;
        if(Child2 >= Array<DATATYPE> :: GetSize())
        {
            if ( Child1 >= Array<DATATYPE> :: GetSize())
            {
                Array<DATATYPE> :: Get(Hole) = tdt;
                return ( dt );
            }
            if(Array<DATATYPE> :: Get ( Child1 ) < tdt)
            {
                Array<DATATYPE> :: Get(Hole) = Array<DATATYPE> :: Get(Child1);
                Array<DATATYPE> :: Get(Child1) = tdt;
                return(dt);
            }
            Array<DATATYPE> :: Get(Hole) = tdt;
            return ( dt );
        }
        Pick = Child1;
        if(Array<DATATYPE> :: Get ( Child1 ) > Array<DATATYPE> :: Get(Child2))
            Pick = Child2;
        if(Array<DATATYPE> :: Get ( Pick ) < tdt)
        {
            Array<DATATYPE> :: Get ( Hole) = Array<DATATYPE> :: Get( Pick );
            Pick = Hole;
        }
        else
        {
            Array<DATATYPE> :: Get(Hole) = tdt;
            return ( dt );
        }
    }
    return ( dt );
}

template < class DATATYPE>
DATATYPE PQueueOfPtr<DATATYPE> :: Push(DATATYPE &Value)
{
    unsigned int Index;
    ///unsigned int Base;
    DATATYPE tdt;

    Array<DATATYPE> :: Add(Value);

    Index = Array<DATATYPE>::GetSize();
    --Index;
    while(Index)
    {
        if(Array<DATATYPE>::Get(Index) < (*Array<DATATYPE>::Get((Index-1)>>1)))

```

```

    dt = Array<DATATYPE> :: Get(Index);
    Array<DATATYPE>::Get(Index) = Array<DATATYPE>::Get((Index-1)>>1);
    Array<DATATYPE>::Get((Index-1)>>1) = tdt;
    nd-x = (Index-1) >> 1;

```

```

    break;
}
return ( dt );
}

```

```

template < class DATATYPE>
DATATYPE QueueOfPtr<DATATYPE> :: Pop()
{
    DATATYPE tdt;
    //DATATYPE tdt2;
    //int Index;
    unsigned int sorted;
    unsigned int Pick;
    unsigned int Child1;
    unsigned int Child2;
    unsigned int Hole;

    if( !Array<DATATYPE>::GetSize() )
        return ( dt ); // error

    dt = Array<DATATYPE> :: Get ( 0 );

    if( Array<DATATYPE> :: GetSize() == 1 )
    {
        Array<DATATYPE> :: RemoveAt ( 0 );
        return ( dt );
    }

    dt = Array<DATATYPE> :: Get ( Array<DATATYPE> :: GetSize() - 1 );
    Array<DATATYPE> :: RemoveAt ( Array<DATATYPE> :: GetSize() - 1 );
    Hole =
        sorted = 0,
        while( !sorted )
        {
            Child1 = 2 * Hole + 1;
            Child2 = 2 * Hole + 2;
            if( Child2 >= Array<DATATYPE> :: GetSize() )
            {
                if( Child1 >= Array<DATATYPE> :: GetSize() )
                {
                    Array<DATATYPE> :: Get(Hole) = tdt;
                    return ( dt );
                }
                if( Array<DATATYPE> :: Get ( Child1 ) < (*tdt) )
                {
                    Array<DATATYPE> :: Get(Hole) = Array<DATATYPE> :: Get(Child1);
                    Array<DATATYPE> :: Get(Child1) = tdt;
                    return(dt);
                }
            }
            Array<DATATYPE> :: Get(Hole) = tdt;
            return ( dt );
        }

    Child1 = Child1;
    if( Array<DATATYPE> :: Get ( Child1 ) > (*Array<DATATYPE> :: Get(Child2)))
        Child1 = Child2;
    if( Array<DATATYPE> :: Get ( Pick ) < (*tdt) )
    {
        Array<DATATYPE> :: Get ( Hole) = Array<DATATYPE> :: Get( Pick );
        Hole = Pick;
    }

    Array<DATATYPE> :: Get(Hole) = tdt;
    return ( dt );
}

return ( dt );
}

```





```

        Priority=NormalPriority, char* originatorAETitle=NULL,
        UINT16 originatorMessageID=0x0000);
    DICOMViewLog& dvl, DICOMDatabase& dtb);
    DICOMViewLog& dvl, DICOMDatabase& dtb, CallBackObject& cbo);
    virtual ~User();

private:
    bool IsCancelled(PDU_Service& PDU, UINT16 mid);
    bool IsCancelled(ApplicationEntity& ae, UINT ae_index);
    bool Cancel(PDU_Service& PDU, UINT16 mid);
    bool Echo(ApplicationEntity& AE);
    bool Echo(PDU_Service& PDU);
    bool Find(ApplicationEntity &AE, DICOMDataObject& ddo_mask,
        Array<DICOMDataObject> & ddo_found,
        BYTE root, UINT16 Priority=NormalPriority);
    bool Find(PDU_Service &PDU, DICOMDataObject& DDO,
        Array<DICOMDataObject> & found_list,
        BYTE root, UINT16 Priority=NormalPriority);
    bool Find(ApplicationEntity &AE, DICOMDataObject& ddo_mask,
        BYTE root, UINT16 Priority=NormalPriority);
    bool Find(PDU_Service &PDU, DICOMDataObject& DDO,
        Array<DICOMDataObject> & found_list,
        BYTE root, UINT16 Priority=NormalPriority);
    bool Make(ApplicationEntity &AE, DICOMDataObject& ddo_mask,
        char* destAET, BYTE root, UINT16 Priority=NormalPriority);
    bool Make(PDU_Service &PDU, DICOMDataObject& DDO,
        char* SOP, char* destAET, UINT16 Priority=NormalPriority);
    bool VerifyCommandAndID(DICOMCommandObject& DCO,
        UINT16 Command, UINT16 MessageID);
    bool VerifyAndVerify(PDU_Service &PDU, DICOMCommandObject &DCO,
        DICOMDataObject &DDO, UINT16 Command,
        UINT16 MessageID, const char* service);
};

class DICOMServiceClass : public ServiceClass
{
public:
    DICOMServiceClass(PDU_Service& PDU, DICOMCommandObject& DCO);
    DICOMServiceClass(PDU_Service &PDU, DICOMCommandObject &DCO, UINT16* cancelledID);
    DICOMServiceClass(PDU_Service &PDU, DICOMCommandObject &DCO, UINT16* cancelledID);
    DICOMServiceClass(PDU_Service &PDU, DICOMCommandObject &DCO, UINT16* cancelledID);
    DICOMServiceClass(PDU_Service &PDU, DICOMCommandObject &DCO);
    bool CancelAndProcess(PDU_Service &PDU, DICOMCommandObject& DCO, UINT16* cancelledID);

    DICOMViewLog& dvl, DICOMDatabase& dtb);
    DICOMViewLog& dvl, DICOMDatabase& dtb, CallBackObject& cbo);
    virtual ~DICOMServiceClass();

private:
    void GetData(DICOMCommandObject& DCO);
};

#endif // !defined(_SERVICE_CLASS_H_INCLUDED_)

```

```

/*****
*
* Copyright (c) 2000, Louisiana State University, School of Medicine
*
* This DICOM object library was developed from the University of California,
* Berkeley MDCOM Network Transport Libraries, in full compliance
* with the license access licence and copyright note below. The DICOM object
* library however contains conceptual deviations from the UCDCM library,
* as well as important bug and performance fixes, and CANNOT be
* redistributed/modified without our permission.
*
* Technical Contact: oleg@bit.csc.lsu.edu
*
*****/
#ifdef _WIN32
#include 
#endif
#ifdef _WIN64
#include 
#endif

/* Utility functions */
UINT ZeroMem(BYTE* mem, UINT Count) { memset((void *) mem, 0, Count); }
inline void ZeroMem(char* mem, UINT Count) { memset((void *) mem, 0, Count); }
void Set2DBufferY(BYTE* buf, UINT32 buf_size,
                  UINT32 nrows);
bool CreateAndSetCurrentDirectory(char *dir);
bool GetDICOMSubstring(char* str, char*sub);
bool SetDICOMSubstring(char* str, char*sub);
bool SerializeString(FILE* fp, char* str, bool is_loading);
bool SerializeBool(FILE* fp, bool& x, bool is_loading);
bool SerializeBOOL(FILE* fp, BOOL& x, bool is_loading);
bool SerializeBYTE(FILE* fp, BYTE& x, bool is_loading);
bool SerializeInteger(FILE* fp, int& x, bool is_loading);
bool SerializeUINT(FILE* fp, UINT& x, bool is_loading);
bool SerializeDouble(FILE* fp, double& x, bool is_loading);
inline bool IsEmptyString(char* s)
{
    if(s==NULL) return true;
    if(s[0]==0) return true;
    return false;
};
inline bool IsUniqueString(char* s)
{
    if(IsEmptyString(s)) return false;
    if(strchr(s, '*') || strchr(s, '?') ||
       strchr(s, '\\')) return false;
    return true;
};
inline bool IsBlankString(char* s)
{
    if(IsEmptyString(s)) return true;
    for(UINT i=0; i<strlen(s); i++)
    {
        if(s[i]!=' ') return false;
    }
    return true;
};
BOOL Copy(BYTE *, BYTE *, UINT);
char* ShortFileName(char* fullpathname);
char* DICOMSubstring(BYTE* str, UINT32 str_len,
                     char* sub, UINT32 sub_len, int number=1);
char* DICOMSubstring(char* str, char* sub, UINT32 sub_len,
                     int number=1);
int Char(char* str, char c);
int CharReverse(char* str, char c);
UINT StrLength(BYTE *);
long Length(char* filename);

#endif

```

```

/*****
*
* Copyright (c) 2000, Louisiana State University, School of Medicine
*
* This object library was developed based on University of California,
* Davis DICOM Network Transport Libraries, in full compliance
* with the copyright note below. This version however contains conceptual
* details from the UCDMC library, as well as important bug and performance
* fixes. It cannot be used/copied/distributed without our permission
*
* For contact: oleg@bit.csc.lsu.edu
*
*****/
#ifdef __H_INCLUDED_
#define VERSION_H_INCLUDED_

/* Version related information
*
* This file contains the version number/Class that will be embedded not
* only in the executable, but in the default PDU Service transfer Class.
*
*/
# define IMPLEMENTATION_CLASS_STRING "1.2.10008.1968"
#ifdef __WIN32_
# define IMPLEMENTATION_VERSION_STRING "2.0_DCM_WIN32"
#else
#ifdef __VXWORKS_
# define IMPLEMENTATION_VERSION_STRING "2.0_DCM_UMIPS"
#else
#ifdef __MIPS_
# define IMPLEMENTATION_VERSION_STRING "2.0_DCM_SL5"
#else
#ifdef __IA64_
# define IMPLEMENTATION_VERSION_STRING "2.0_DCM_MAC"
#else
#ifdef __SUNOS_
# define IMPLEMENTATION_VERSION_STRING "2.0_DCM_SUNOS"
#else
# define IMPLEMENTATION_VERSION_STRING "2.0_DCM_OTHER"
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif

```

```

/*****
 *
 * Copyright (C) 2000, Louisiana State University, School of Medicine
 *
 * This object library was developed based on University of California,
 * San Diego DICOM Network Transport Libraries, in full compliance
 * with the copyright note below. This version however contains conceptual
 * changes from the UCDCM library, as well as important bug and performance
 * improvements. It cannot be used/copied/distributed without our permission
 *
 * Contact: oleg@bit.csc.lsu.edu
 *
 *****/
# "application.h"

```

```

/*****
 *
 * Application Context Class
 *
 *****/
ApplicationContext :: ApplicationContext()
{
    mType = 0x10;
    mReserved1 = 0;
    mLength = 0;
}

ApplicationContext :: ApplicationContext(UID &uid)
{
    mType = 0x10;
    mReserved1 = 0;
    mLength = 0;
    ApplicationContextName = uid;
}

ApplicationContext :: ApplicationContext(BYTE *name)
{
    mType = 0x10;
    mReserved1 = 0;
    mLength = 0;
    ApplicationContextName.Set(name);
}

ApplicationContext :: ~ApplicationContext()
{
    // Do not de-allocate specifically
}

void ApplicationContext :: Set(UID &uid)
{
    ApplicationContextName = uid;
}

void ApplicationContext :: Set(BYTE *name)
{
    ApplicationContextName.Set(name);
}

bool ApplicationContext :: Write(Buffer &Link)
{
    mLength = Size();
    mType = 0x10;
    mReserved1 = 0;
    mLength = Length;
    Link.Write((BYTE *) ApplicationContextName.GetBuffer(), Length);
    return TRUE;
}

bool ApplicationContext :: Read(Buffer &Link)
{
    mType = 0x10;
    mReserved1 = 0;
    mLength = 0;
    Link.ReadDynamic(Link);
    return TRUE;
}

bool ApplicationContext :: ReadDynamic(Buffer &Link)
{
    mType = 0x10;
    mReserved1 = 0;
    mLength = 0;
    Link.ReadDynamic((BYTE *) ApplicationContextName.GetBuffer(), Length);
    ApplicationContextName.GetBuffer()[Length] = '\0';
}

```

```

    AbstractSyntaxName.SetLength(Length);
    return TRUE;
}

UINT16 AbstractSyntaxName::Size()
{
    return (AbstractSyntaxName.GetSize();
    sizeof(BYTE) + sizeof(BYTE) + sizeof(UINT16) + Length );
}

/*****
 * AbstractSyntax Class
 *****/
AbstractSyntax : AbstractSyntax()
{
    mLength = 30;
    mReserved1 = 0;
    mLink = 0;
}

AbstractSyntax::AbstractSyntax(UINT &uid)
{
    mLength = 30;
    mReserved1 = 0;
    mLink = 0;
    AbstractSyntaxName = uid;
}

AbstractSyntax::AbstractSyntax(BYTE *name)
{
    mLength = 30;
    mReserved1 = 0;
    mLink = 0;
    AbstractSyntaxName.Set(name);
}

AbstractSyntax::~AbstractSyntax()
{
    // Nothing to de-allocate specifically
}

void AbstractSyntax::Set(UINT &uid)
{
    AbstractSyntaxName = uid;
}

void AbstractSyntax::Set(BYTE *name)
{
    AbstractSyntaxName.Set(name);
}

BOOL AbstractSyntax::Write(Buffer &Link)
{
    mLength = Size();
    mLink = Link;
    mReserved1 = 0;
    mLink = 0;
    AbstractSyntaxName.GetBuffer(), Length);
    return TRUE;
}

BOOL AbstractSyntax::Read(Buffer &Link)
{
    mLink = Link;
    mReserved1 = 0;
    mLink = 0;
    AbstractSyntaxName.GetBuffer(), Length);
    AbstractSyntaxName.GetBuffer()[Length] = '\0';
    AbstractSyntaxName.SetLength(Length);
    return TRUE;
}

UINT16 AbstractSyntax::Size()
{
    return AbstractSyntaxName.GetSize();
}

```

```

        sizeof(BYTE) + sizeof(BYTE) + sizeof(UINT16) + Length );
    }

/*****
 *
 * TransferSyntax
 *
 *****/
TransferSyntax::TransferSyntax()
{
    mType = 0x40;
    mReserved1 = 0;
    mLength = 0;
}

TransferSyntax::TransferSyntax(UID &uid)
{
    mType = 0x40;
    mReserved1 = 0;
    mLength = 0;
    mTransferSyntaxName = uid;
}

TransferSyntax::TransferSyntax(BYTE *name)
{
    mType = 0x40;
    mReserved1 = 0;
    mLength = 0;
    mTransferSyntaxName.Set(name);
}

TransferSyntax::~TransferSyntax()
{
    // Do not de-allocate specifically
}

void TransferSyntax::Set(UID &uid)
{
    mTransferSyntaxName = uid;
}

void TransferSyntax::Set(BYTE *name)
{
    mTransferSyntaxName.Set(name);
}

bool TransferSyntax::Write(Buffer &Link)
{
    mTransferSyntaxName.GetSize();
    mType = 0x40;
    mReserved1 = 0;
    mLength = 0;
    mTransferSyntaxName.GetBuffer(), Length);
    mTransferSyntaxName.GetLength();
    mTransferSyntaxName.SetLength(Length);
    mTransferSyntaxName.SetLength(Length);
}

bool TransferSyntax::Read(Buffer &Link)
{
    mType = 0x40;
    mReserved1 = 0;
    mLength = 0;
    mTransferSyntaxName.ReadDynamic(Link);
}

bool TransferSyntax::ReadDynamic(Buffer &Link)
{
    mType = 0x40;
    mReserved1 = 0;
    mLength = 0;
    mTransferSyntaxName.GetBuffer(), Length);
    mTransferSyntaxName.GetBuffer()[Length] = '\0';
    mTransferSyntaxName.SetLength(Length);
    mTransferSyntaxName.SetLength(Length);
}

UINT16 TransferSyntax::Size()
{
    mTransferSyntaxName.GetSize();
    sizeof(BYTE) + sizeof(BYTE) + sizeof(UINT16) + Length );
}

/*****
 *
 *****/

```

```

* Implementation Class
*
****/
ImplementationClass :: ImplementationClass()
{
    mType = x52;
    mLength = 0;
    mImplementationName.Set((BYTE*)IMPLEMENTATION_CLASS_STRING);
}
ImplementationClass :: ImplementationClass(UID &uid)
{
    mType = x52;
    mLength = 0;
    mImplementationName = uid;
}
ImplementationClass :: ImplementationClass(BYTE *name)
{
    mType = x52;
    mLength = 0;
    mImplementationName.Set(name);
}
ImplementationClass :: ~ImplementationClass()
{
    // de-allocate specifically
};

void ImplementationClass :: Set(UID &uid)
{
    mImplementationName = uid;
}

void ImplementationClass :: Set(BYTE *name)
{
    mImplementationName.Set(name);
}

BOOLEAN ImplementationClass :: Write(Buffer &Link)
{
    mLength = mImplementationName.GetSize();
    mType = x52;
    mLength = mImplementationName.GetSize();
    mImplementationName.SetBuffer((BYTE *)ImplementationName.GetBuffer(), Length);
    Link.Write(mType, Length);
}

BOOLEAN ImplementationClass :: Read(Buffer &Link)
{
    mType = x52;
    mLength = Link.ReadDynamic(Length);
}

BOOLEAN ImplementationClass :: ReadDynamic(Buffer &Link)
{
    mLength = Link.ReadDynamic(Length);
    mType = x52;
    mImplementationName.SetBuffer((BYTE *)ImplementationName.GetBuffer(), Length);
    mImplementationName.GetBuffer()[Length] = '\0';
    mImplementationName.SetLength(Length);
}

UINT ImplementationClass :: Size()
{
    return mImplementationName.GetSize();
}

// Implementation Version
ImplementationVersion :: ImplementationVersion()
{

```





```

}
PresentationContext :: PresentationContext( AbstractSyntax &Abs,
                                           TransferSyntax &Tran )
{
    m_Abs = &Abs;
    m_Trans = &Tran;
    m_Count = 0;
    m_PresentationContextID = uniq8odd();
    m_Reserved1 = 0;
    m_Reserved2 = 0;
    m_Reserved3 = 0;
    m_Reserved4 = 0;
}
PresentationContext :: ~PresentationContext()
{
}

void PresentationContext :: SetAbstractSyntax( AbstractSyntax &Abs )
{
    m_Abs = &Abs;
}

void PresentationContext :: AddTransferSyntax( TransferSyntax &Tran )
{
    m_Trans = &Tran;
}

BOOL PresentationContext :: Write ( Buffer &Link )
{
    m_Count++;
    m_Type = TYPE_PRESENTATION_CONTEXT;
    m_Reserved1 = 0;
    m_Length = 0;
    m_PresentationContextID;
    m_Reserved2 = 0;
    m_Reserved3 = 0;
    m_Reserved4 = 0;
    m_Count = Write( Link );
    m_Count--;
    if ( m_Count < TrnSyntax.GetSize() )
    {
        m_Link[ m_Index ].Write( Link );
        m_Index++;
    }
    return TRUE;
}

BOOL PresentationContext :: Read ( Buffer &Link )
{
    m_Count = 0;
    m_Type = 0;
    m_Count = ReadDynamic( Link );
}

BOOL PresentationContext :: ReadDynamic ( Buffer &Link )
{
    m_Count = 0;
    m_Trans = 0;

    m_Reserved1 = 0;
    m_Length = 0;
    m_PresentationContextID;
    m_Reserved2 = 0;
    m_Reserved3 = 0;
    m_Reserved4 = 0;

    m_Length = sizeof( BYTE ) - sizeof( BYTE ) - sizeof( BYTE ) - sizeof( BYTE );
    m_Count = Read( Link );
    m_Count -= AbsSyntax.Size();
    if ( m_Count > 0 )
    {
        m_Link = 0;
        m_Count = Tran.Size();
        m_Count = Add ( Tran );
    }
    return TRUE;
}

```











```

{
    ...
    ReadDynamic(Link));
}

BOOL CInfo::ReadDynamic(Buffer &Link)
{
    ...
    Count;
    TempByte;
    ... *PresContext;

    ...
    Version;
    ...
    ... * CalledApTitle, 16);
    ... * CallingApTitle, 16);
    ... * Reserved3, 32);
    ...[16] = '\0';
    ...[16] = '\0';

    ... - sizeof(UINT16) - sizeof(UINT16) - 16 - 16 - 32;
    ...

    ... (BYTE *) &TempByte, sizeof(BYTE));
    ... (TempByte)

    ... 0x50: // User information
    ... Info.ReadDynamic(Link);
    ... nt = Count - UserInfo.Size() - UserInfo.UserInfoBaggage;
    ...
    ... 0x20: // Presentation context
    ... PresContext = new PresentationContext;
    ... PresContext->TrnSyntax.ClearType = TRUE;
    ... PresContext->ReadDynamic(Link);
    ... nt = Count - PresContext->Size();
    ... PresContexts.Add(*PresContext);
    ... PresContext->TrnSyntax.ClearType = FALSE;
    ... delete PresContext;
    ...
    ... 0x10: // Application context
    ... Context.ReadDynamic(Link);
    ... nt = Count - AppContext.Size();
    ...
    ... // Unknown item
    ... Kill(Count-1);
    ... nt = -1;

    ... return TRUE;
}

UINT CInfo::Size()
{
    ... (UINT16) + sizeof(UINT16) + 16 + 16 + 32;
    ... PresContext.Size();

    ... PresContexts.GetSize())
    ... PresContexts[Index].Size();

    ... Info.Size();
    ... + sizeof(BYTE) + sizeof(BYTE) + sizeof(UINT32) );
}

```





# **APPENDIX 1**

**Inventor: Pianykh, et al.**

**Title: Radiologist Workstation**

**Atty. Doc. #6451.064**

**Source Code Listing (Volume 2)**

```

/*
 * @author Imtiaz Hossain
 *
 * @version 2.0
 *
 * Date : 10/05/00
 *
 * Header file for JPEGlib
 */

#include<iostream.h>
#include <afxwin.h>

#ifdef __cplusplus
extern "C" {
#endif // __cplusplus
#include <time.h>
#include "jpeglib.h"
#ifdef __cplusplus
}
#endif // __cplusplus

class CodecErrorHandler
{
public:
    int n_ErrorFlag;

    // Methods

    int GetFlag(){
        return n_ErrorFlag;
    }
    void SetFlag(int flag){
        n_ErrorFlag=flag;
    }

    void Notify_stderr(int flag){
        cout <<"Error: #" <<flag<<endl;
    }
}

class JPEG: public CodecErrorHandler
{
public:

    // static consts
    // Error definitions
    static const int SUCCESS;
    static const int MEMORY_ALLOC_ERROR;
    static const int FILE_READ_ERROR;
    static const int FILE_WRITE_ERROR;
    static const int JPEGLIB_STRUCT_INIT_ERROR;

    // Coding types

    static const int DEFAULT_CODING;
    static const int BASELINE;
    static const int PROGRESSIVE;
    static const int LOSSLESS;

    // Resolution
    static const int DEFAULT_RES;
    static const int ONE_BYTE;
    static const int TWO_BYTE;
    static const int THREE_BYTE;
    static const int FOUR_BYTE;

```

```
// Color Planes
static const int DEFAULT_BPP;
static const int Gray;
static const int RGB;
```

```
// Image Quality
static const int DEF_QUALITY;
```

```
// Compression Ratio
static const float DEF_RATIO;
```

```
// Time limit on the compression
static const long DEF_TIME;
```

```
// regular public variables.
```

```
int n_ImageHeight;
int n_ImageWidth;
int n_ImageBpp;
int n_ImageResolution;
int n_ImageCodingType;
int n_ImageQuality;
```

```
// Constructor (Default)
JPEG(){}
```

```
// Encoder Methods
```

```
/* The EncoderParam method loads the values for
```

- 1) The Height of the image.
- 2) The Width of the image.
- 3) The number of color components per pixel, i.e. color depth.
- 4) [Optional] Image resolution. Number of bits per pixel.  
This is usually a Grayscale thing. But the comprehension of the input data buffer (in parameter 1) will change. We usually have two options 8 or 12. DICOM will support upto 16.
- 5) [Optional] The type of Encoding. ... i.e. BASELINE, PROGRESSIVE, etc. etc. Again this might call for a change in the way the decoder produces the output stream of BYTES.
- 6) [Optional] Quality of the compression. On a scale of [0 - 100] this parameter determines the tradeoff between compression and image quality. For larger values of this parameter, we might actually have a blow-up of the size instead of compression. A 100 on this parameter does NOT imply lossless compression.

```
*/
```

```
void EncoderParam(int n_Height, int n_Width, int n_Bpp, int n_Quality=DEF_QUALITY, int n_Coding=DEFAULT_CODING, int n_Res=DEFAULT_RES);
```

```
/* The Encode method works on the values fed into the EncoderParam method. These values eventually show up in the modified "IJG" structure for jpeg compression. The compression proceeds as per these values. The compressed buffer is returned by the function. The only input parameter is a pointer to an "int", so as to avoid having to declare a "struct". The total length of the returned buffer is written into this variable.
```

```
*/
```

```
BYTE * Encode(BYTE *jpeg_get_Buffer, int *length, int *ret_quality, int n_Height, int n_Width, int n_Bpp, int n_Quality=DEF_QUALITY, int n_Res=DEFAULT_RES, int n_Coding=DEFAULT_CODING);
```

```
BYTE * Encode(BYTE *jpeg_get_Buffer, int *length, int *ret_quality, int n_Height, int n_Width, int n_Bpp, float n_Ratio=DEF_RATIO, long n_Time=DEF_TIME, int n_Res=DEFAULT_RES, int n_Coding=DEFAULT_CODING);
```

```
/* The decoder needs to have certain information about its input before it processes it. The input to the decoder is in the form of a stream of BYTES consisting of the compressed data. The height, width and number of colors of the uncompressed image need to be specified. The pixel-resolution also needs to be specified. The DecodeParam method just serves to collect these values before the decoding starts.
```

```
*/
```

```

/* The Decode method decompresses the input buffer specified in the method DecodeParam and
   returns the uncompressed data as a stream of BYTES in the form R,G,B,R,G,B.... or a
   sequence of Gray values as the case may be.
*/

BYTE * Decode(BYTE *CompressedBuffer, int length);

BYTE * ChopWindow(BYTE * whole_stream, int* height, int* width, int bpp);

BYTE * Resolution_Convertor(BYTE *jpeg_get_Buffer, int *bpp, int *res, int Height,int *Width);

};

```

```

/*
 * jpegint.h
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file provides common declarations for the various JPEG modules.
 * These declarations are considered internal to the JPEG library; most
 * applications using the library shouldn't need to include this file.
 */

/* Declarations for both compression & decompression */

typedef enum {
    JBUF_PASS_THRU, /* Plain stripwise operation */
    /* Remaining modes require a full-image buffer to have been created */
    JBUF_SAVE_SOURCE, /* Run source subobject only, save output */
    JBUF_CRANK_DEST, /* Run dest subobject only, using saved data */
    JBUF_SAVE_AND_PASS /* Run both subobjects, save output */
} J_BUF_MODE;

/* Values of global_state field (jdapi.c has some dependencies on ordering!) */
#define CSTATE_START 100 /* after create_compress */
#define CSTATE_SCANNING 101 /* start_compress done, write_scanlines OK */
#define CSTATE_RAW_OK 102 /* start_compress done, write_raw_data OK */
#define CSTATE_WRCOEFS 103 /* jpeg_write_coefficients done */
#define DSTATE_START 200 /* after create_decompress */
#define DSTATE_INHEADER 201 /* reading header markers, no SOS yet */
#define DSTATE_READY 202 /* found SOS, ready for start_decompress */
#define DSTATE_PRELOAD 203 /* reading multiscan file in start_decompress */
#define DSTATE_PRESCAN 204 /* performing dummy pass for 2-pass quant */
#define DSTATE_SCANNING 205 /* start_decompress done, read_scanlines OK */
#define DSTATE_RAW_OK 206 /* start_decompress done, read_raw_data OK */
#define DSTATE_BUFIMAGE 207 /* expecting jpeg_start_output */
#define DSTATE_BUFPOST 208 /* looking for SOS/EOI in jpeg_finish_output */
#define DSTATE_RDCOEFS 209 /* reading file in jpeg_read_coefficients */
#define DSTATE_STOPPING 210 /* looking for EOI in jpeg_finish_decompress */

/* Declarations for compression modules */

/* Master control module */
struct jpeg_comp_master {
    JMETHOD(void, prepare_for_pass, (j_compress_ptr cinfo));
    JMETHOD(void, pass_startup, (j_compress_ptr cinfo));
    JMETHOD(void, finish_pass, (j_compress_ptr cinfo));

    /* State variables made visible to other modules */
    boolean call_pass_startup; /* True if pass_startup must be called */
    boolean is_last_pass; /* True during last pass */
};

/* Main buffer control (downsampled-data buffer) */
struct jpeg_c_main_controller {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo, J_BUF_MODE pass_mode));
    JMETHOD(void, process_data, (j_compress_ptr cinfo,
        JSAMPARRAY input_buf, JDIMENSION *in_row_ctr,
        JDIMENSION in_rows_avail));
};

/* Compression preprocessing (downsampling input buffer control) */
struct jpeg_c_prep_controller {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo, J_BUF_MODE pass_mode));
    JMETHOD(void, pre_process_data, (j_compress_ptr cinfo,
        JSAMPARRAY input_buf,
        JDIMENSION *in_row_ctr,
        JDIMENSION in_rows_avail,
        JSAMPIMAGE output_buf,
        JDIMENSION *out_row_group_ctr,
        JDIMENSION out_row_groups_avail));
};

/* Coefficient buffer control */
struct jpeg_c_coef_controller {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo, J_BUF_MODE pass_mode));
    JMETHOD(void, compress_data, (j_compress_ptr cinfo,
        JSAMPIMAGE input_buf));
};

```

```

/* Colorspace conversion */
struct jpeg_color_converter {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo));
    JMETHOD(void, color_convert, (j_compress_ptr cinfo,
        JSAMPARRAY input_buf, JSAMPIMAGE output_buf,
        JDIMENSION output_row, int num_rows));
};

/* Downsampling */
struct jpeg_downsampler {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo));
    JMETHOD(void, downsample, (j_compress_ptr cinfo,
        JSAMPIMAGE input_buf, JDIMENSION in_row_index,
        JSAMPIMAGE output_buf,
        JDIMENSION out_row_group_index));

    boolean need_context_rows; /* TRUE if need rows above & below */
};

/* Forward DCT (also controls coefficient quantization) */
struct jpeg_forward_dct {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo));
    /* perhaps this should be an array??? */
    JMETHOD(void, forward_DCT, (j_compress_ptr cinfo,
        jpeg_component_info * compptr,
        JSAMPARRAY sample_data, JBLOCKROW coef_blocks,
        JDIMENSION start_row, JDIMENSION start_col,
        JDIMENSION num_blocks));
};

/* Entropy encoding */
struct jpeg_entropy_encoder {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo, boolean gather_statistics));
    JMETHOD(boolean, encode_mcu, (j_compress_ptr cinfo, JBLOCKROW *MCU_data));
    JMETHOD(void, finish_pass, (j_compress_ptr cinfo));
};

/* Marker writing */
struct jpeg_marker_writer {
    JMETHOD(void, write_file_header, (j_compress_ptr cinfo));
    JMETHOD(void, write_frame_header, (j_compress_ptr cinfo));
    JMETHOD(void, write_scan_header, (j_compress_ptr cinfo));
    JMETHOD(void, write_file_trailer, (j_compress_ptr cinfo));
    JMETHOD(void, write_tables_only, (j_compress_ptr cinfo));
    /* These routines are exported to allow insertion of extra markers */
    /* Probably only COM and APPn markers should be written this way */
    JMETHOD(void, write_marker_header, (j_compress_ptr cinfo, int marker,
        unsigned int datalen));
    JMETHOD(void, write_marker_byte, (j_compress_ptr cinfo, int val));
};

/* Declarations for decompression modules */

/* Master control module */
struct jpeg_decomp_master {
    JMETHOD(void, prepare_for_output_pass, (j_decompress_ptr cinfo));
    JMETHOD(void, finish_output_pass, (j_decompress_ptr cinfo));

    /* State variables made visible to other modules */
    boolean is_dummy_pass; /* True during 1st pass for 2-pass quant */
};

/* Input control module */
struct jpeg_input_controller {
    JMETHOD(int, consume_input, (j_decompress_ptr cinfo));
    JMETHOD(void, reset_input_controller, (j_decompress_ptr cinfo));
    JMETHOD(void, start_input_pass, (j_decompress_ptr cinfo));
    JMETHOD(void, finish_input_pass, (j_decompress_ptr cinfo));

    /* State variables made visible to other modules */
    boolean has_multiple_scans; /* True if file has multiple scans */
    boolean eoi_reached; /* True when EOI has been consumed */
};

/* Main buffer control (downsampled-data buffer) */
struct jpeg_d_main_controller {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo, J_BUF_MODE pass_mode));
    JMETHOD(void, process_data, (j_decompress_ptr cinfo,

```

```

        JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
        JDIMENSION out_rows_avail));
};

/* Coefficient buffer control */
struct jpeg_d_coef_controller {
    JMETHOD(void, start_input_pass, (j_decompress_ptr cinfo));
    JMETHOD(int, consume_data, (j_decompress_ptr cinfo));
    JMETHOD(void, start_output_pass, (j_decompress_ptr cinfo));
    JMETHOD(int, decompress_data, (j_decompress_ptr cinfo,
        JSAMPIMAGE output_buf));
    /* Pointer to array of coefficient virtual arrays, or NULL if none */
    jvirt_barray_ptr *coef_arrays;
};

/* Decompression postprocessing (color quantization buffer control) */
struct jpeg_d_post_controller {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo, J_BUF_MODE pass_mode));
    JMETHOD(void, post_process_data, (j_decompress_ptr cinfo,
        JSAMPIMAGE input_buf,
        JDIMENSION *in_row_group_ctr,
        JDIMENSION in_row_groups_avail,
        JSAMPARRAY output_buf,
        JDIMENSION *out_row_ctr,
        JDIMENSION out_rows_avail));
};

/* Marker reading & parsing */
struct jpeg_marker_reader {
    JMETHOD(void, reset_marker_reader, (j_decompress_ptr cinfo));
    /* Read markers until SOS or EOI.
     * Returns same codes as are defined for jpeg_consume_input:
     * JPEG_SUSPENDED, JPEG_REACHED_SOS, or JPEG_REACHED_EOI.
     */
    JMETHOD(int, read_markers, (j_decompress_ptr cinfo));
    /* Read a restart marker --- exported for use by entropy decoder only */
    jpeg_marker_parser_method read_restart_marker;

    /* State of marker reader --- nominally internal, but applications
     * supplying COM or APPn handlers might like to know the state.
     */
    boolean saw_SOI; /* found SOI? */
    boolean saw_SOF; /* found SOF? */
    int next_restart_num; /* next restart number expected (0-7) */
    unsigned int discarded_bytes; /* # of bytes skipped looking for a marker */
};

/* Entropy decoding */
struct jpeg_entropy_decoder {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo));
    JMETHOD(boolean, decode_mcu, (j_decompress_ptr cinfo,
        JBLOCKROW *MCU_data));

    /* This is here to share code between baseline and progressive decoders; */
    /* other modules probably should not use it */
    boolean insufficient_data; /* set TRUE after emitting warning */
};

/* Inverse DCT (also performs dequantization) */
typedef JMETHOD(void, inverse_DCT_method_ptr,
    (j_decompress_ptr cinfo, jpeg_component_info * comp_ptr,
    JCOEFPTR coef_block,
    JSAMPARRAY output_buf, JDIMENSION output_col));

struct jpeg_inverse_dct {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo));
    /* It is useful to allow each component to have a separate IDCT method. */
    inverse_DCT_method_ptr inverse_DCT[MAX_COMPONENTS];
};

/* Upsampling (note that upsampler must also call color converter) */
struct jpeg_upsampler {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo));
    JMETHOD(void, upsample, (j_decompress_ptr cinfo,
        JSAMPIMAGE input_buf,
        JDIMENSION *in_row_group_ctr,
        JDIMENSION in_row_groups_avail,
        JSAMPARRAY output_buf,
        JDIMENSION *out_row_ctr,
        JDIMENSION out_rows_avail));
};

```

```

    boolean need_context_rows;    /* TRUE if need rows above & below */
};

/* Colorspace conversion */
struct jpeg_color_deconverter {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo));
    JMETHOD(void, color_convert, (j_decompress_ptr cinfo,
        JSAMPIMAGE input_buf, JDIMENSION input_row,
        JSAMPARRAY output_buf, int num_rows));
};

/* Color quantization or color precision reduction */
struct jpeg_color_quantizer {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo, boolean is_pre_scan));
    JMETHOD(void, color_quantize, (j_decompress_ptr cinfo,
        JSAMPARRAY input_buf, JSAMPARRAY output_buf,
        int num_rows));
    JMETHOD(void, finish_pass, (j_decompress_ptr cinfo));
    JMETHOD(void, new_color_map, (j_decompress_ptr cinfo));
};

/* Miscellaneous useful macros */

#undef MAX
#define MAX(a,b)    ((a) > (b) ? (a) : (b))
#undef MIN
#define MIN(a,b)    ((a) < (b) ? (a) : (b))

/* We assume that right shift corresponds to signed division by 2 with
 * rounding towards minus infinity. This is correct for typical "arithmetic
 * shift" instructions that shift in copies of the sign bit. But some
 * C compilers implement >> with an unsigned shift. For these machines you
 * must define RIGHT_SHIFT_IS_UNSIGNED.
 * RIGHT_SHIFT provides a proper signed right shift of an INT32 quantity.
 * It is only applied with constant shift counts. SHIFT_TEMPS must be
 * included in the variables of any routine using RIGHT_SHIFT.
 */
#ifdef RIGHT_SHIFT_IS_UNSIGNED
#define SHIFT_TEMPS INT32 shift_temp;
#define RIGHT_SHIFT(x,shft) \
    ((shift_temp = (x)) < 0 ? \
        (shift_temp >> (shft)) | ((~(INT32) 0) << (32-(shft))) : \
        (shift_temp >> (shft)))
#else
#define SHIFT_TEMPS
#define RIGHT_SHIFT(x,shft) ((x) >> (shft))
#endif

/* Short forms of external names for systems with brain-damaged linkers. */
#ifdef NEED_SHORT_EXTERNAL_NAMES
#define jinit_compress_master    jICompress
#define jinit_c_master_control  jICMaster
#define jinit_c_main_controller jICMainC
#define jinit_c_prep_controller jICPrepC
#define jinit_c_coef_controller jICCoeFC
#define jinit_color_converter   jICColor
#define jinit_downsampler        jIDownsampler
#define jinit_forward_dct        jIFDCT
#define jinit_huff_encoder        jIHEncoder
#define jinit_phuff_encoder       jIPHEncoder
#define jinit_marker_writer      jIMWriter
#define jinit_master_decompress  jIDMaster
#define jinit_d_main_controller  jIDMainC
#define jinit_d_coef_controller  jIDCoeFC
#define jinit_d_post_controller  jIDPostC
#define jinit_input_controller   jIInCtlr
#define jinit_marker_reader      jIMReader
#define jinit_huff_decoder        jIHDecoder
#define jinit_phuff_decoder       jIPHDecoder
#define jinit_inverse_dct        jIIDCT
#define jinit_upsampler          jIUpsampler
#define jinit_color_deconverter  jIDColor
#define jinit_1pass_quantizer     jI1Quant
#define jinit_2pass_quantizer     jI2Quant

```



```

#define jinit_merged_upsampler jIMUpsampler
#define jinit_memory_mgr jIMemMgr
#define jdiv_round_up jDivRound
#define jround_up jRound
#define jcopy_sample_rows jCopySamples
#define jcopy_block_row jCopyBlocks
#define jzero_far jZeroFar
#define jpeg_zigzag_order jZIGTable
#define jpeg_natural_order jZAGTable
#endif /* NEED_SHORT_EXTERNAL_NAMES */

/* Compression module initialization routines */
EXTERN(void) jinit_compress_master JPP((j_compress_ptr cinfo));
EXTERN(void) jinit_c_master_control JPP((j_compress_ptr cinfo,
    boolean transcode_only));
EXTERN(void) jinit_c_main_controller JPP((j_compress_ptr cinfo,
    boolean need_full_buffer));
EXTERN(void) jinit_c_prep_controller JPP((j_compress_ptr cinfo,
    boolean need_full_buffer));
EXTERN(void) jinit_c_coef_controller JPP((j_compress_ptr cinfo,
    boolean need_full_buffer));
EXTERN(void) jinit_color_converter JPP((j_compress_ptr cinfo));
EXTERN(void) jinit_downsampler JPP((j_compress_ptr cinfo));
EXTERN(void) jinit_forward_dct JPP((j_compress_ptr cinfo));
EXTERN(void) jinit_huff_encoder JPP((j_compress_ptr cinfo));
EXTERN(void) jinit_phuff_encoder JPP((j_compress_ptr cinfo));
EXTERN(void) jinit_marker_writer JPP((j_compress_ptr cinfo));
/* Decompression module initialization routines */
EXTERN(void) jinit_master_decompress JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_d_main_controller JPP((j_decompress_ptr cinfo,
    boolean need_full_buffer));
EXTERN(void) jinit_d_coef_controller JPP((j_decompress_ptr cinfo,
    boolean need_full_buffer));
EXTERN(void) jinit_d_post_controller JPP((j_decompress_ptr cinfo,
    boolean need_full_buffer));
EXTERN(void) jinit_input_controller JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_marker_reader JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_huff_decoder JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_phuff_decoder JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_inverse_dct JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_upsampler JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_color_deconverter JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_1pass_quantizer JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_2pass_quantizer JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_merged_upsampler JPP((j_decompress_ptr cinfo));
/* Memory manager initialization */
EXTERN(void) jinit_memory_mgr JPP((j_common_ptr cinfo));

/* Utility routines in jutils.c */
EXTERN(long) jdiv_round_up JPP((long a, long b));
EXTERN(long) jround_up JPP((long a, long b));
EXTERN(void) jcopy_sample_rows JPP((JSAMPARRAY input_array, int source_row,
    JSAMPARRAY output_array, int dest_row,
    int num_rows, JDIMENSION num_cols));
EXTERN(void) jcopy_block_row JPP((JBLOCKROW input_row, JBLOCKROW output_row,
    JDIMENSION num_blocks));
EXTERN(void) jzero_far JPP((void * target, size_t bytestozero));
/* Constant tables in jutils.c */
#if 0 /* This table is not actually needed in v6a */
extern const int jpeg_zigzag_order[]; /* natural coef order to zigzag order */
#endif
extern const int jpeg_natural_order[]; /* zigzag coef order to natural order */

/* Suppress undefined-structure complaints if necessary. */

#ifdef INCOMPLETE_TYPES_BROKEN
#ifdef AM_MEMORY_MANAGER /* only jmemmgr.c defines these */
struct jvirt_sarray_control { long dummy; };
struct jvirt_barray_control { long dummy; };
#endif
#endif /* INCOMPLETE_TYPES_BROKEN */

```

```

/*
 * jpeglib.h
 *
 * Copyright (C) 1991-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file defines the application interface for the JPEG library.
 * Most applications using the library need only include this file,
 * and perhaps jerror.h if they want to know the exact error codes.
 */

#ifndef JPEGLIB_H
#define JPEGLIB_H

/*
 * First we include the configuration files that record how this
 * installation of the JPEG library is set up.  jconfig.h can be
 * generated automatically for many systems.  jmorecfg.h contains
 * manual configuration options that most people need not worry about.
 */

#ifndef JCONFIG_INCLUDED /* in case jinclude.h already did */
#include "jconfig.h" /* widely used configuration options */
#endif
#include "jmorecfg.h" /* seldom changed options */

#ifndef STDIO_H
#include <stdio.h>
#define STDIO_H
#endif

/* Version ID for the JPEG library.
 * Might be useful for tests like "#if JPEG_LIB_VERSION >= 60".
 */
#define JPEG_LIB_VERSION 62 /* Version 6b */

/* Various constants determining the sizes of things.
 * All of these are specified by the JPEG standard, so don't change them
 * if you want to be compatible.
 */
#define DCTSIZE 8 /* The basic DCT block is 8x8 samples */
#define DCTSIZE2 64 /* DCTSIZE squared; # of elements in a block */
#define NUM_QUANT_TBLS 4 /* Quantization tables are numbered 0..3 */
#define NUM_HUFF_TBLS 4 /* Huffman tables are numbered 0..3 */
#define NUM_ARITH_TBLS 16 /* Arith-coding tables are numbered 0..15 */
#define MAX_COMPS_IN_SCAN 4 /* JPEG limit on # of components in one scan */
#define MAX_SAMP_FACTOR 4 /* JPEG limit on sampling factors */
/* Unfortunately, some bozo at Adobe saw no reason to be bound by the standard;
 * the PostScript DCT filter can emit files with many more than 10 blocks/MCU.
 * If you happen to run across such a file, you can up D_MAX_BLOCKS_IN_MCU
 * to handle it. We even let you do this from the jconfig.h file. However,
 * we strongly discourage changing C_MAX_BLOCKS_IN_MCU; just because Adobe
 * sometimes emits noncompliant files doesn't mean you should too.
 */
#define C_MAX_BLOCKS_IN_MCU 10 /* compressor's limit on blocks per MCU */
#ifndef D_MAX_BLOCKS_IN_MCU
#define D_MAX_BLOCKS_IN_MCU 10 /* decompressor's limit on blocks per MCU */
#endif

/* Data structures for images (arrays of samples and of DCT coefficients).
 * On 80x86 machines, the image arrays are too big for near pointers,
 * but the pointer arrays can fit in near memory.
 */
typedef JSAMPLE *JSAMPROW; /* ptr to one image row of pixel samples. */
typedef JSAMPROW *JSAMPARRAY; /* ptr to some rows (a 2-D sample array) */
typedef JSAMPARRAY *JSAMPIMAGE; /* a 3-D sample array: top index is color */

typedef JCOEF JBLOCK[DCTSIZE2]; /* one block of coefficients */
typedef JBLOCK *JBLOCKROW; /* pointer to one row of coefficient blocks */
typedef JBLOCKROW *JBLOCKARRAY; /* a 2-D array of coefficient blocks */
typedef JBLOCKARRAY *JBLOCKIMAGE; /* a 3-D array of coefficient blocks */

typedef JCOEF *JCOEFPTR; /* useful in a couple of places */

```

```

/* Types for JPEG compression parameters and working tables. */

/* DCT coefficient quantization tables. */

typedef struct {
    /* This array gives the coefficient quantizers in natural array order
     * (not the zigzag order in which they are stored in a JPEG DQT marker).
     * CAUTION: IJG versions prior to v6a kept this array in zigzag order.
     */
    UINT16 quantval[DCTSIZE2]; /* quantization step for each coefficient */
    /* This field is used only during compression. It's initialized FALSE when
     * the table is created, and set TRUE when it's been output to the file.
     * You could suppress output of a table by setting this to TRUE.
     * (See jpeg_suppress_tables for an example.)
     */
    boolean sent_table; /* TRUE when table has been output */
} JQUANT_TBL;

/* Huffman coding tables. */

typedef struct {
    /* These two fields directly represent the contents of a JPEG DHT marker */
    UINT8 bits[17]; /* bits[k] = # of symbols with codes of k */
    /* length k bits; bits[0] is unused */
    UINT8 huffval[256]; /* The symbols, in order of incr code length */
    /* This field is used only during compression. It's initialized FALSE when
     * the table is created, and set TRUE when it's been output to the file.
     * You could suppress output of a table by setting this to TRUE.
     * (See jpeg_suppress_tables for an example.)
     */
    boolean sent_table; /* TRUE when table has been output */
} JHUFF_TBL;

/* Basic info about one component (color channel). */

typedef struct {
    /* These values are fixed over the whole image. */
    /* For compression, they must be supplied by parameter setup; */
    /* for decompression, they are read from the SOF marker. */
    int component_id; /* identifier for this component (0..255) */
    int component_index; /* its index in SOF or cinfo->comp_info[] */
    int h_samp_factor; /* horizontal sampling factor (1..4) */
    int v_samp_factor; /* vertical sampling factor (1..4) */
    int quant_tbl_no; /* quantization table selector (0..3) */
    /* These values may vary between scans. */
    /* For compression, they must be supplied by parameter setup; */
    /* for decompression, they are read from the SOS marker. */
    /* The decompressor output side may not use these variables. */
    int dc_tbl_no; /* DC entropy table selector (0..3) */
    int ac_tbl_no; /* AC entropy table selector (0..3) */

    /* Remaining fields should be treated as private by applications. */

    /* These values are computed during compression or decompression startup: */
    /* Component's size in DCT blocks.
     * Any dummy blocks added to complete an MCU are not counted; therefore
     * these values do not depend on whether a scan is interleaved or not.
     */
    JDIMENSION width_in_blocks;
    JDIMENSION height_in_blocks;
    /* Size of a DCT block in samples. Always DCTSIZE for compression.
     * For decompression this is the size of the output from one DCT block,
     * reflecting any scaling we choose to apply during the IDCT step.
     * Values of 1,2,4,8 are likely to be supported. Note that different
     * components may receive different IDCT scalings.
     */
    int DCT_scaled_size;
    /* The downsampled dimensions are the component's actual, unpadded number
     * of samples at the main buffer (preprocessing/compression interface), thus
     * downsampled_width = ceil(image_width * H1/Hmax)
     * and similarly for height. For decompression, IDCT scaling is included, so
     * downsampled_width = ceil(image_width * H1/Hmax * DCT_scaled_size/DCTSIZE)
     */
    JDIMENSION downsampled_width; /* actual width in samples */
    JDIMENSION downsampled_height; /* actual height in samples */
    /* This flag is used only for decompression. In cases where some of the

```

```

    * components will be ignored (eg grayscale output from YCbCr image),
    * we can skip most computations for the unused components.
    */
    boolean component_needed; /* do we need the value of this component? */

    /* These values are computed before starting a scan of the component. */
    /* The decompressor output side may not use these variables. */
    int MCU_width; /* number of blocks per MCU, horizontally */
    int MCU_height; /* number of blocks per MCU, vertically */
    int MCU_blocks; /* MCU_width * MCU_height */
    int MCU_sample_width; /* MCU width in samples, MCU_width*DCT_scaled_size */
    int last_col_width; /* # of non-dummy blocks across in last MCU */
    int last_row_height; /* # of non-dummy blocks down in last MCU */

    /* Saved quantization table for component; NULL if none yet saved.
     * See jdinput.c comments about the need for this information.
     * This field is currently used only for decompression.
     */
    JQUANT_TBL * quant_table;

    /* Private per-component storage for DCT or IDCT subsystem. */
    void * dct_table;
} jpeg_component_info;

/* The script for encoding a multiple-scan file is an array of these: */
typedef struct {
    int comps_in_scan; /* number of components encoded in this scan */
    int component_index[MAX_COMPS_IN_SCAN]; /* their SOF/comp_info[] indexes */
    int Ss, Se; /* progressive JPEG spectral selection parms */
    int Ah, Al; /* progressive JPEG successive approx. parms */
} jpeg_scan_info;

/* The decompressor can save APPn and COM markers in a list of these: */
typedef struct jpeg_marker_struct * jpeg_saved_marker_ptr;

struct jpeg_marker_struct {
    jpeg_saved_marker_ptr next; /* next in list, or NULL */
    UINT8 marker; /* marker code: JPEG_COM, or JPEG_APP0+n */
    unsigned int original_length; /* # bytes of data in the file */
    unsigned int data_length; /* # bytes of data saved at data[] */
    JOCTET * data; /* the data contained in the marker */
    /* the marker length word is not counted in data_length or original_length */
}

/* Known color spaces. */
typedef enum {
    JCS_UNKNOWN, /* error/unspecified */
    JCS_GRAYSCALE, /* monochrome */
    JCS_RGB, /* red/green/blue */
    JCS_YCbCr, /* Y/Cb/Cr (also known as YUV) */
    JCS_CMYK, /* C/M/Y/K */
    JCS_YCCK, /* Y/Cb/Cr/K */
} J_COLOR_SPACE;

/* DCT/IDCT algorithm options. */
typedef enum {
    JDCT_ISLOW, /* slow but accurate integer algorithm */
    JDCT_IFAST, /* faster, less accurate integer method */
    JDCT_FLOAT, /* floating-point: accurate, fast on fast HW */
} J_DCT_METHOD;

#ifndef JDCT_DEFAULT /* may be overridden in jconfig.h */
#define JDCT_DEFAULT JDCT_ISLOW
#endif
#ifndef JDCT_FASTEST /* may be overridden in jconfig.h */
#define JDCT_FASTEST JDCT_IFAST
#endif

/* Dithering options for decompression. */
typedef enum {
    JDITHER_NONE, /* no dithering */
    JDITHER_ORDERED, /* simple ordered dither */
    JDITHER_FS, /* Floyd-Steinberg error diffusion dither */
} J_DITHER_MODE;

```

```

/* Common fields between JPEG compression and decompression master structs. */

#define jpeg_common_fields \
    struct jpeg_error_mgr * err; /* Error handler module */\
    struct jpeg_memory_mgr * mem; /* Memory manager module */\
    struct jpeg_progress_mgr * progress; /* Progress monitor, or NULL if none */\
    void * client_data; /* Available for use by application */\
    boolean is_decompressor; /* So common code can tell which is which */\
    int global_state /* For checking call sequence validity */

/* Routines that are to be used by both halves of the library are declared
 * to receive a pointer to this structure. There are no actual instances of
 * jpeg_common_struct, only of jpeg_compress_struct and jpeg_decompress_struct.
 */
struct jpeg_common_struct {
    jpeg_common_fields; /* Fields common to both master struct types */
    /* Additional fields follow in an actual jpeg_compress_struct or
     * jpeg_decompress_struct. All three structs must agree on these
     * initial fields! (This would be a lot cleaner in C++. )
     */
};

typedef struct jpeg_common_struct * j_common_ptr;
typedef struct jpeg_compress_struct * j_compress_ptr;
typedef struct jpeg_decompress_struct * j_decompress_ptr;

/* Master record for a compression instance */
struct jpeg_compress_struct {
    jpeg_common_fields; /* Fields shared with jpeg_decompress_struct */
    /* Destination for compressed data */
    struct jpeg_destination_mgr * dest;
    /* Description of source image --- these fields must be filled in by
     * outer application before starting compression. in_color_space must
     * be correct before you can even call jpeg_set_defaults().
     */
    JDIMENSION image_width; /* input image width */
    JDIMENSION image_height; /* input image height */
    int input_components; /* # of color components in input image */
    J_COLOR_SPACE in_color_space; /* colorspace of input image */
    double input_gamma; /* image gamma of input image */
    /* Compression parameters --- these fields must be set before calling
     * jpeg_start_compress(). We recommend calling jpeg_set_defaults() to
     * initialize everything to reasonable defaults, then changing anything
     * the application specifically wants to change. That way you won't get
     * burnt when new parameters are added. Also note that there are several
     * helper routines to simplify changing parameters.
     */
    /* ### I added this index variable as a counter to the compressed buffer.*/
    int index;
    int data_precision; /* bits of precision in image data */
    int num_components; /* # of color components in JPEG image */
    J_COLOR_SPACE jpeg_color_space; /* colorspace of JPEG image */
    jpeg_component_info * comp_info;
    /* comp_info[i] describes component that appears i'th in SOF */
    JQUANT_TBL * quant_tbl_ptrs[NUM_QUANT_TBLS];
    /* ptrs to coefficient quantization tables, or NULL if not defined */
    JHUFF_TBL * dc_huff_tbl_ptrs[NUM_HUFF_TBLS];
    JHUFF_TBL * ac_huff_tbl_ptrs[NUM_HUFF_TBLS];
    /* ptrs to Huffman coding tables, or NULL if not defined */
    UINT8 arith_dc_L[NUM_ARITH_TBLS]; /* L values for DC arith-coding tables */
    UINT8 arith_dc_U[NUM_ARITH_TBLS]; /* U values for DC arith-coding tables */
    UINT8 arith_ac_K[NUM_ARITH_TBLS]; /* Kx values for AC arith-coding tables */
    int num_scans; /* # of entries in scan_info array */

```

```

const jpeg_scan_info * scan_info; /* script for multi-scan file, or NULL */
/* The default value of scan_info is NULL, which causes a single-scan
 * sequential JPEG file to be emitted. To create a multi-scan file,
 * set num_scans and scan_info to point to an array of scan definitions.
 */

boolean raw_data_in;      /* TRUE=caller supplies downsampled data */
boolean arith_code;       /* TRUE=arithmetic coding, FALSE=Huffman */
boolean optimize_coding; /* TRUE=optimize entropy encoding parms */
boolean CCIR601_sampling; /* TRUE=first samples are cosited */
int smoothing_factor;     /* 1..100, or 0 for no input smoothing */
J_DCT_METHOD dct_method; /* DCT algorithm selector */

/* The restart interval can be specified in absolute MCUs by setting
 * restart_interval, or in MCU rows by setting restart_in_rows
 * (in which case the correct restart_interval will be figured
 * for each scan).
 */
unsigned int restart_interval; /* MCUs per restart, or 0 for no restart */
int restart_in_rows;          /* if > 0, MCU rows per restart interval */

/* Parameters controlling emission of special markers. */

boolean write_JFIF_header; /* should a JFIF marker be written? */
UINT8 JFIF_major_version; /* What to write for the JFIF version number */
UINT8 JFIF_minor_version;
/* These three values are not used by the JPEG code, merely copied */
/* into the JFIF APP0 marker. density_unit can be 0 for unknown, */
/* 1 for dots/inch, or 2 for dots/cm. Note that the pixel aspect */
/* ratio is defined by X_density/Y_density even when density_unit=0. */
UINT8 density_unit; /* JFIF code for pixel size units */
UINT16 X_density; /* Horizontal pixel density */
UINT16 Y_density; /* Vertical pixel density */
boolean write_Adobe_marker; /* should an Adobe marker be written? */

/* State variable: index of next scanline to be written to
 * jpeg_write_scanlines(). Application may use this to control its
 * processing loop, e.g., "while (next_scanline < image_height)".
 */
JDIMENSION next_scanline; /* 0 .. image_height-1 */

/* Remaining fields are known throughout compressor, but generally
 * should not be touched by a surrounding application.
 */

/* These fields are computed during compression startup
 */
boolean progressive_mode; /* TRUE if scan script uses progressive mode */
int max_h_samp_factor; /* largest h_samp_factor */
int max_v_samp_factor; /* largest v_samp_factor */

JDIMENSION total_iMCU_rows; /* # of iMCU rows to be input to coef ctlr */
/* The coefficient controller receives data in units of MCU rows as defined
 * for fully interleaved scans (whether the JPEG file is interleaved or not).
 * There are v_samp_factor * DCTSIZE sample rows of each component in an
 * "iMCU" (interleaved MCU) row.
 */

/*
 * These fields are valid during any one scan.
 * They describe the components and MCUs actually appearing in the scan.
 */
int comps_in_scan; /* # of JPEG components in this scan */
jpeg_comp_info * cur_comp_info[MAX_COMPS_IN_SCAN];
/* *cur_comp_info[i] describes component that appears i'th in SOS */

JDIMENSION MCUs_per_row; /* # of MCUs across the image */
JDIMENSION MCU_rows_in_scan; /* # of MCU rows in the image */

int blocks_in_MCU; /* # of DCT blocks per MCU */
int MCU_membership[C_MAX_BLOCKS_IN_MCU];
/* MCU_membership[i] is index in cur_comp_info of component owning */
/* i'th block in an MCU */

int Ss, Se, Ah, Al; /* progressive JPEG parameters for scan */

/*
 * Links to compression subobjects (methods and private variables of modules)
 */

```

```

    */
    struct jpeg_comp_master * master;
    struct jpeg_c_main_controller * main;
    struct jpeg_c_prep_controller * prep;
    struct jpeg_c_coef_controller * coef;
    struct jpeg_marker_writer * marker;
    struct jpeg_color_converter * cconvert;
    struct jpeg_downsampler * downsampler;
    struct jpeg_forward_dct * fdct;
    struct jpeg_entropy_encoder * entropy;
    jpeg_scan_info * script_space; /* workspace for jpeg_simple_progression */
    int script_space_size;
};

/* Master record for a decompression instance */

struct jpeg_decompress_struct {
    jpeg_common_fields; /* Fields shared with jpeg_compress_struct */

    /* Source of compressed data */
    struct jpeg_source_mgr * src;

    /* Basic description of image --- filled in by jpeg_read_header(). */
    /* Application may inspect these values to decide how to process image. */

    JDIMENSION image_width; /* nominal image width (from SOF marker) */
    JDIMENSION image_height; /* nominal image height */
    int num_components; /* # of color components in JPEG image */
    J_COLOR_SPACE jpeg_color_space; /* colorspace of JPEG image */

    /* Decompression processing parameters --- these fields must be set before
    * calling jpeg_start_decompress(). Note that jpeg_read_header() initializes
    * them to default values.
    */

    J_COLOR_SPACE out_color_space; /* colorspace for output */

    unsigned int scale_num, scale_denom; /* fraction by which to scale image */

    double output_gamma; /* image gamma wanted in output */

    boolean buffered_image; /* TRUE=multiple output passes */
    boolean raw_data_out; /* TRUE=downsampled data wanted */

    J_DCT_METHOD dct_method; /* IDCT algorithm selector */
    boolean do_fancy_upsampling; /* TRUE=apply fancy upsampling */
    boolean do_block_smoothing; /* TRUE=apply interblock smoothing */

    boolean quantize_colors; /* TRUE=colormapped output wanted */
    /* the following are ignored if not quantize_colors: */
    J_DITHER_MODE dither_mode; /* type of color dithering to use */
    boolean two_pass_quantize; /* TRUE=use two-pass color quantization */
    int desired_number_of_colors; /* max # colors to use in created colormap */
    /* these are significant only in buffered-image mode: */
    boolean enable_1pass_quant; /* enable future use of 1-pass quantizer */
    boolean enable_external_quant; /* enable future use of external colormap */
    boolean enable_2pass_quant; /* enable future use of 2-pass quantizer */

    /* Description of actual output image that will be returned to application.
    * These fields are computed by jpeg_start_decompress().
    * You can also use jpeg_calc_output_dimensions() to determine these values
    * in advance of calling jpeg_start_decompress().
    */

    JDIMENSION output_width; /* scaled image width */
    JDIMENSION output_height; /* scaled image height */
    int out_color_components; /* # of color components in out_color_space */
    int output_components; /* # of color components returned */
    /* output_components is 1 (a colormap index) when quantizing colors;
    * otherwise it equals out_color_components.
    */
    int rec_outbuf_height; /* min recommended height of scanline buffer */
    /* If the buffer passed to jpeg_read_scanlines() is less than this many rows
    * high, space and time will be wasted due to unnecessary data copying.
    * Usually rec_outbuf_height will be 1 or 2, at most 4.
    */

    /* When quantizing colors, the output colormap is described by these fields.
    * The application can supply a colormap by setting colormap non-NULL before

```

```

* calling jpeg_start_decompress; otherwise a colormap is created during
* jpeg_start_decompress or jpeg_start_output.
* The map has out_color_components rows and actual_number_of_colors columns.
*/
int actual_number_of_colors; /* number of entries in use */
JSAMPARRAY colormap; /* The color map as a 2-D pixel array */

/* State variables: these variables indicate the progress of decompression.
* The application may examine these but must not modify them.
*/

/* Row index of next scanline to be read from jpeg_read_scanlines().
* Application may use this to control its processing loop, e.g.,
* "while (output_scanline < output_height)".
*/
JDIMENSION output_scanline; /* 0 .. output_height-1 */

/* Current input scan number and number of iMCU rows completed in scan.
* These indicate the progress of the decompressor input side.
*/
int input_scan_number; /* Number of SOS markers seen so far */
JDIMENSION input_iMCU_row; /* Number of iMCU rows completed */

/* The "output scan number" is the notional scan being displayed by the
* output side. The decompressor will not allow output scan/row number
* to get ahead of input scan/row, but it can fall arbitrarily far behind.
*/
int output_scan_number; /* Nominal scan number being displayed */
JDIMENSION output_iMCU_row; /* Number of iMCU rows read */

/* Current progression status. coef_bits[c][i] indicates the precision
* with which component c's DCT coefficient i (in zigzag order) is known.
* It is -1 when no data has yet been received, otherwise it is the point
* transform (shift) value for the most recent scan of the coefficient
* (thus, 0 at completion of the progression).
* This pointer is NULL when reading a non-progressive file.
*/
int (*coef_bits)[DCTSIZE2]; /* -1 or current Al value for each coef */

/* Internal JPEG parameters --- the application usually need not look at
* these fields. Note that the decompressor output side may not use
* any parameters that can change between scans.
*/

/* Quantization and Huffman tables are carried forward across input
* datastreams when processing abbreviated JPEG datastreams.
*/
JQUANT_TBL * quant_tbl_ptrs[NUM_QUANT_TBLS];
/* ptrs to coefficient quantization tables, or NULL if not defined */
JHUFF_TBL * dc_huff_tbl_ptrs[NUM_HUFF_TBLS];
JHUFF_TBL * ac_huff_tbl_ptrs[NUM_HUFF_TBLS];
/* ptrs to Huffman coding tables, or NULL if not defined */

/* These parameters are never carried across datastreams, since they
* are given in SOF/SOS markers or defined to be reset by SOI.
*/

int data_precision; /* bits of precision in image data */

jpeg_component_info * comp_info;
/* comp_info[i] describes component that appears i'th in SOF */

boolean progressive_mode; /* TRUE if SOFn specifies progressive mode */
boolean arith_code; /* TRUE=arithmetic coding, FALSE=Huffman */

UINT8 arith_dc_L[NUM_ARITH_TBLS]; /* L values for DC arith-coding tables */
UINT8 arith_dc_U[NUM_ARITH_TBLS]; /* U values for DC arith-coding tables */
UINT8 arith_ac_K[NUM_ARITH_TBLS]; /* Kx values for AC arith-coding tables */

unsigned int restart_interval; /* MCUs per restart interval, or 0 for no restart */

/* These fields record data obtained from optional markers recognized by
* the JPEG library.
*/
boolean saw_JFIF_marker; /* TRUE iff a JFIF APP0 marker was found */
/* Data copied from JFIF marker; only valid if saw_JFIF_marker is TRUE: */
UINT8 JFIF_major_version; /* JFIF version number */
UINT8 JFIF_minor_version;

```



```

UINT8 density_unit;      /* JFIF code for pixel size units */
UINT16 X_density;        /* Horizontal pixel density */
UINT16 Y_density;        /* Vertical pixel density */
boolean saw_Adobe_marker; /* TRUE iff an Adobe APP14 marker was found */
UINT8 Adobe_transform;   /* Color transform code from Adobe marker */

boolean CCIR601_sampling; /* TRUE=first samples are cosited */

/* Aside from the specific data retained from APPn markers known to the
 * library, the uninterpreted contents of any or all APPn and COM markers
 * can be saved in a list for examination by the application.
 */
jpeg_saved_marker_ptr marker_list; /* Head of list of saved markers */

/* Remaining fields are known throughout decompressor, but generally
 * should not be touched by a surrounding application.
 */

/*
 * These fields are computed during decompression startup
 */
int max_h_samp_factor; /* largest h_samp_factor */
int max_v_samp_factor; /* largest v_samp_factor */

int min_DCT_scaled_size; /* smallest DCT_scaled_size of any component */

JDIMENSION total_iMCU_rows; /* # of iMCU rows in image */
/* The coefficient controller's input and output progress is measured in
 * units of "iMCU" (interleaved MCU) rows. These are the same as MCU rows
 * in fully interleaved JPEG scans, but are used whether the scan is
 * interleaved or not. We define an iMCU row as v_samp_factor DCT block
 * rows of each component. Therefore, the IDCT output contains
 * v_samp_factor*DCT_scaled_size sample rows of a component per iMCU row.
 */
JSAMPLE * sample_range_limit; /* table for fast range-limiting */

/*
 * These fields are valid during any one scan.
 * They describe the components and MCUs actually appearing in the scan.
 * Note that the decompressor output side must not use these fields.
 */
int comps_in_scan; /* # of JPEG components in this scan */
jpeg_component_info * cur_comp_info[MAX_COMPS_IN_SCAN];
/* *cur_comp_info[i] describes component that appears i'th in SOS */

JDIMENSION MCUs_per_row; /* # of MCUs across the image */
JDIMENSION MCU_rows_in_scan; /* # of MCU rows in the image */

int blocks_in_MCU; /* # of DCT blocks per MCU */
int MCU_membership[D_MAX_BLOCKS_IN_MCU];
/* MCU_membership[i] is index in cur_comp_info of component owning */
/* i'th block in an MCU */

int Ss, Se, Ah, Al; /* progressive JPEG parameters for scan */

/* This field is shared between entropy decoder and marker parser.
 * It is either zero or the code of a JPEG marker that has been
 * read from the data source, but has not yet been processed.
 */
int unread_marker;

/*
 * Links to decompression subobjects (methods, private variables of modules)
 */
struct jpeg_decomp_master * master;
struct jpeg_d_main_controller * main;
struct jpeg_d_coef_controller * coef;
struct jpeg_d_post_controller * post;
struct jpeg_input_controller * inputctl;
struct jpeg_marker_reader * marker;
struct jpeg_entropy_decoder * entropy;
struct jpeg_inverse_dct * idct;
struct jpeg_upsampler * upsample;
struct jpeg_color_deconverter * cconvert;
struct jpeg_color_quantizer * cquantize;
};

/* "Object" declarations for JPEG modules that may be supplied or called

```

```

* directly by the surrounding application.
* As with all objects in the JPEG library, these structs only define the
* publicly visible methods and state variables of a module. Additional
* private fields may exist after the public ones.
*/

```

```

/* Error handler object */

```

```

struct jpeg_error_mgr {
    /* Error exit handler: does not return to caller */
    JMETHOD(void, error_exit, (j_common_ptr cinfo));
    /* Conditionally emit a trace or warning message */
    JMETHOD(void, emit_message, (j_common_ptr cinfo, int msg_level));
    /* Routine that actually outputs a trace or error message */
    JMETHOD(void, output_message, (j_common_ptr cinfo));
    /* Format a message string for the most recent JPEG error or message */
    JMETHOD(void, format_message, (j_common_ptr cinfo, char * buffer));
#define JMSG_LENGTH_MAX 200 /* recommended size of format_message buffer */
    /* Reset error state variables at start of a new image */
    JMETHOD(void, reset_error_mgr, (j_common_ptr cinfo));

```

```

    /* The message ID code and any parameters are saved here.
     * A message can have one string parameter or up to 8 int parameters.
     */

```

```

    int msg_code;
#define JMSG_STR_PARM_MAX 80
    union {
        int i[8];
        char s[JMSG_STR_PARM_MAX];
    } msg_parm;

```

```

    /* Standard state variables for error facility */

```

```

    int trace_level; /* max msg_level that will be displayed */

```

```

    /* For recoverable corrupt-data errors, we emit a warning message,
     * but keep going unless emit_message chooses to abort. emit_message
     * should count warnings in num_warnings. The surrounding application
     * can check for bad data by seeing if num_warnings is nonzero at the
     * end of processing.
     */

```

```

    long num_warnings; /* number of corrupt-data warnings */

```

```

    /* These fields point to the table(s) of error message strings.
     * An application can change the table pointer to switch to a different
     * message list (typically, to change the language in which errors are
     * reported). Some applications may wish to add additional error codes
     * that will be handled by the JPEG library error mechanism; the second
     * table pointer is used for this purpose.
     */

```

```

    /* First table includes all errors generated by JPEG library itself.
     * Error code 0 is reserved for a "no such error string" message.
     */

```

```

    const char * const * jpeg_message_table; /* Library errors */
    int last_jpeg_message; /* Table contains strings 0..last_jpeg_message */
    /* Second table can be added by application (see cjpeg/djpeg for example).
     * It contains strings numbered first_addon_message..last_addon_message.
     */

```

```

    const char * const * addon_message_table; /* Non-library errors */
    int first_addon_message; /* code for first string in addon table */
    int last_addon_message; /* code for last string in addon table */
};

```

```

/* Progress monitor object */

```

```

struct jpeg_progress_mgr {
    JMETHOD(void, progress_monitor, (j_common_ptr cinfo));

    long pass_counter; /* work units completed in this pass */
    long pass_limit; /* total number of work units in this pass */
    int completed_passes; /* passes completed so far */
    int total_passes; /* total number of passes expected */
};

```

```

/* Data destination object for compression */

```

```

/* ### Imtiaz: The linked list structure to store the buffer data. */

```

```

typedef struct linked_list
{
    JOCTET *buffer;
    struct linked_list *next;
} buffer_list;

- struct jpeg_destination_mgr {
    JOCTET * next_output_byte; /* => next byte to write in buffer */
    size_t free_in_buffer; /* # of byte spaces remaining in buffer */

    JOCTET *outbuffer;

    buffer_list *head_ptr;
    buffer_list *current_ptr;

    JMETHOD(void, init_destination, (j_compress_ptr cinfo));
    JMETHOD(boolean, empty_output_buffer, (j_compress_ptr cinfo));
    JMETHOD(void, term_destination, (j_compress_ptr cinfo));
- };

- /* Data source object for decompression */

struct jpeg_source_mgr {
    const JOCTET * next_input_byte; /* => next byte to read from buffer */
    size_t bytes_in_buffer; /* # of bytes remaining in buffer */

    JSAMPLE *inbuffer; /* ### Imtiaz: I added this */
    JSAMPLE *head_inbuffer; /* " */

    int buffer_length;

    JMETHOD(void, init_source, (j_decompress_ptr cinfo));
    JMETHOD(boolean, fill_input_buffer, (j_decompress_ptr cinfo));
    JMETHOD(void, skip_input_data, (j_decompress_ptr cinfo, long num_bytes));
    JMETHOD(boolean, resync_to_restart, (j_decompress_ptr cinfo, int desired));
    JMETHOD(void, term_source, (j_decompress_ptr cinfo));
}

/* Memory manager object.
 * Allocates "small" objects (a few K total), "large" objects (tens of K),
 * and "really big" objects (virtual arrays with backing store if needed).
 * The memory manager does not allow individual objects to be freed; rather,
 * each created object is assigned to a pool, and whole pools can be freed
 * at once. This is faster and more convenient than remembering exactly what
 * to free, especially where malloc()/free() are not too speedy.
 * NB: alloc routines never return NULL. They exit to error_exit if not
 * successful.
 */

#define JPOOL_PERMANENT 0 /* lasts until master record is destroyed */
#define JPOOL_IMAGE 1 /* lasts until done with image/datastream */
#define JPOOL_NUMPOOLS 2

typedef struct jvirt_sarray_control * jvirt_sarray_ptr;
typedef struct jvirt_barray_control * jvirt_barray_ptr;

struct jpeg_memory_mgr {
    /* Method pointers */
    JMETHOD(void *, alloc_small, (j_common_ptr cinfo, int pool_id,
        size_t sizeofobject));
    JMETHOD(void *, alloc_large, (j_common_ptr cinfo, int pool_id,
        size_t sizeofobject));
    JMETHOD(JSAMPARRAY, alloc_sarray, (j_common_ptr cinfo, int pool_id,
        JDIMENSION samplesperrow,
        JDIMENSION numrows));
    JMETHOD(JBLOCKARRAY, alloc_barray, (j_common_ptr cinfo, int pool_id,
        JDIMENSION blocksperrow,
        JDIMENSION numrows));
    JMETHOD(jvirt_sarray_ptr, request_virt_sarray, (j_common_ptr cinfo,
        int pool_id,
        boolean pre_zero,
        JDIMENSION samplesperrow,
        JDIMENSION numrows,

```

```

        JDIMENSION maxaccess));
JMETHOD(jvirt_barray_ptr, request_virt_barray, (j_common_ptr cinfo,
        int pool_id,
        boolean pre_zero,
        JDIMENSION blocksperrow,
        JDIMENSION numrows,
        JDIMENSION maxaccess));
JMETHOD(void, realize_virt_arrays, (j_common_ptr cinfo));
JMETHOD(JSAMPARRAY, access_virt_sarray, (j_common_ptr cinfo,
        jvirt_sarray_ptr ptr,
        JDIMENSION start_row,
        JDIMENSION num_rows,
        boolean writable));
JMETHOD(JBLOCKARRAY, access_virt_barray, (j_common_ptr cinfo,
        jvirt_barray_ptr ptr,
        JDIMENSION start_row,
        JDIMENSION num_rows,
        boolean writable));
JMETHOD(void, free_pool, (j_common_ptr cinfo, int pool_id));
JMETHOD(void, self_destruct, (j_common_ptr cinfo));

/* Limit on memory allocation for this JPEG object.  (Note that this is
 * merely advisory, not a guaranteed maximum; it only affects the space
 * used for virtual-array buffers.)  May be changed by outer application
 * after creating the JPEG object.
 */
long max_memory_to_use;

/* Maximum allocation request accepted by alloc_large. */
long max_alloc_chunk;
};

/* Routine signature for application-supplied marker processing methods.
 * Need not pass marker code since it is stored in cinfo->unread_marker.
 */
typedef JMETHOD(boolean, jpeg_marker_parser_method, (j_decompress_ptr cinfo));

/* Declarations for routines called by application.
 * The JPP macro hides prototype parameters from compilers that can't cope.
 * Note JPP requires double parentheses.
 */
#ifdef HAVE_PROTOTYPES
#define JPP(arglist)    arglist
#else
#define JPP(arglist)    ()
#endif

/* Short forms of external names for systems with brain-damaged linkers.
 * We shorten external names to be unique in the first six letters, which
 * is good enough for all known systems.
 * (If your compiler itself needs names to be unique in less than 15
 * characters, you are out of luck.  Get a better compiler.)
 */
#ifdef NEED_SHORT_EXTERNAL_NAMES
#define jpeg_std_error        jStdError
#define jpeg_CreateCompress   jCreaCompress
#define jpeg_CreateDecompress jCreaDecompress
#define jpeg_destroy_compress jDestCompress
#define jpeg_destroy_decompress jDestDecompress
#define jpeg_stdio_dest       jStdDest
#define jpeg_buffer_dest      jBufDest
#define stitch_list           S_List
#define jpeg_stdio_src        jStdSrc
#define jpeg_set_defaults     jSetDefaults
#define jpeg_set_colorspace   jSetColorspace
#define jpeg_default_colorspace jDefColorspace
#define jpeg_set_quality       jSetQuality
#define jpeg_set_linear_quality jSetLQuality
#define jpeg_add_quant_table   jAddQuantTable
#define jpeg_quality_scaling   jQualityScaling
#define jpeg_simple_progression jSimProgress
#define jpeg_suppress_tables   jSuppressTables
#define jpeg_alloc_quant_table jAlcQTable
#define jpeg_alloc_huff_table  jAlcHTable
#define jpeg_start_compress    jStrtCompress

```

```

#define jpeg_write_scanlines    jWrtScanlines
#define jpeg_finish_compress    jFinCompress
#define jpeg_write_raw_data     jWrtRawData
#define jpeg_write_marker       jWrtMarker
#define jpeg_write_m_header     jWrtMHeader
#define jpeg_write_m_byte       jWrtMByte
#define jpeg_write_tables       jWrtTables
#define jpeg_read_header        jReadHeader
#define jpeg_start_decompress    jStrtDecompress
#define jpeg_read_scanlines     jReadScanlines
#define jpeg_finish_decompress   jFinDecompress
#define jpeg_read_raw_data      jReadRawData
#define jpeg_has_multiple_scans jHasMultScn
#define jpeg_start_output       jStrtOutput
#define jpeg_finish_output       jFinOutput
#define jpeg_input_complete     jInComplete
#define jpeg_new_colormap       jNewCMap
#define jpeg_consume_input      jConsumeInput
#define jpeg_calc_output_dimensions jCalcDimensions
#define jpeg_save_markers       jSaveMarkers
#define jpeg_set_marker_processor jSetMarker
#define jpeg_read_coefficients  jReadCoefs
#define jpeg_write_coefficients jWrtCoefs
#define jpeg_copy_critical_parameters jCopyCrit
#define jpeg_abort_compress     jAbrtCompress
#define jpeg_abort_decompress   jAbrtDecompress
#define jpeg_abort              jAbort
#define jpeg_destroy            jDestroy
#define jpeg_resync_to_restart  jResyncRestart
#endif /* NEED_SHORT_EXTERNAL_NAMES */

/* Default error-management setup */
EXTERN(struct jpeg_error_mgr *) jpeg_std_error
    JPP((struct jpeg_error_mgr * err));

/* Initialization of JPEG compression objects.
 * jpeg_create_compress() and jpeg_create_decompress() are the exported
 * names that applications should call. These expand to calls on
 * jpeg_CreateCompress and jpeg_CreateDecompress with additional information
 * passed for version mismatch checking.
 * NB: you must set up the error-manager BEFORE calling jpeg_create_xxx.
 */
#define jpeg_create_compress(cinfo) \
    jpeg_CreateCompress((cinfo), JPEG_LIB_VERSION, \
        (size_t) sizeof(struct jpeg_compress_struct))
#define jpeg_create_decompress(cinfo) \
    jpeg_CreateDecompress((cinfo), JPEG_LIB_VERSION, \
        (size_t) sizeof(struct jpeg_decompress_struct))
EXTERN(void) jpeg_CreateCompress JPP((j_compress_ptr cinfo,
    int version, size_t structsize));
EXTERN(void) jpeg_CreateDecompress JPP((j_decompress_ptr cinfo,
    int version, size_t structsize));

/* Destruction of JPEG compression objects */
EXTERN(void) jpeg_destroy_compress JPP((j_compress_ptr cinfo));
EXTERN(void) jpeg_destroy_decompress JPP((j_decompress_ptr cinfo));

/* Standard data source and destination managers: stdio streams. */
/* Caller is responsible for opening the file before and closing after. */

EXTERN(void) jpeg_stdio_dest JPP((j_compress_ptr cinfo, FILE * outfile));

EXTERN(void) jpeg_buffer_dest JPP((j_compress_ptr cinfo));
EXTERN(void) stitch_list JPP((j_compress_ptr cinfo));

EXTERN(void) jpeg_stdio_src JPP((j_decompress_ptr cinfo, FILE * infile));
EXTERN(void) jpeg_buffer_src JPP((j_decompress_ptr cinfo));

/* Default parameter setup for compression */
EXTERN(void) jpeg_set_defaults JPP((j_compress_ptr cinfo));
/* Compression parameter setup aids */
EXTERN(void) jpeg_set_colorspace JPP((j_compress_ptr cinfo,
    J_COLOR_SPACE colorspace));
EXTERN(void) jpeg_default_colorspace JPP((j_compress_ptr cinfo));
EXTERN(void) jpeg_set_quality JPP((j_compress_ptr cinfo, int quality,
    boolean force_baseline));
EXTERN(void) jpeg_set_linear_quality JPP((j_compress_ptr cinfo,
    int scale_factor,

```

```

        boolean force_baseline));
EXTERN(void) jpeg_add_quant_table JPP((j_compress_ptr cinfo, int which_tbl,
        const unsigned int *basic_table,
        int scale_factor,
        boolean force_baseline));
EXTERN(int) jpeg_quality_scaling JPP((int quality));
EXTERN(void) jpeg_simple_progression JPP((j_compress_ptr cinfo));
EXTERN(void) jpeg_suppress_tables JPP((j_compress_ptr cinfo,
        boolean suppress));
EXTERN(JQUANT_TBL *) jpeg_alloc_quant_table JPP((j_common_ptr cinfo));
EXTERN(JHUFF_TBL *) jpeg_alloc_huff_table JPP((j_common_ptr cinfo));

/* Main entry points for compression */
EXTERN(void) jpeg_start_compress JPP((j_compress_ptr cinfo,
        boolean write_all_tables));
EXTERN(JDIMENSION) jpeg_write_scanlines JPP((j_compress_ptr cinfo,
        JSAMPARRAY scanlines,
        JDIMENSION num_lines));
EXTERN(void) jpeg_finish_compress JPP((j_compress_ptr cinfo));

/* Replaces jpeg_write_scanlines when writing raw downsampled data. */
EXTERN(JDIMENSION) jpeg_write_raw_data JPP((j_compress_ptr cinfo,
        JSAMPIMAGE data,
        JDIMENSION num_lines));

/* Write a special marker. See libjpeg.doc concerning safe usage. */
EXTERN(void) jpeg_write_marker
        JPP((j_compress_ptr cinfo, int marker,
        const JOCTET *dataptr, unsigned int datalen));
/* Same, but piecemeal. */
EXTERN(void) jpeg_write_m_header
        JPP((j_compress_ptr cinfo, int marker, unsigned int datalen));
EXTERN(void) jpeg_write_m_byte
        JPP((j_compress_ptr cinfo, int val));

/* Alternate compression function: just write an abbreviated table file */
EXTERN(void) jpeg_write_tables JPP((j_compress_ptr cinfo));

/* Decompression startup: read start of JPEG datastream to see what's there */
EXTERN(int) jpeg_read_header JPP((j_decompress_ptr cinfo,
        boolean require_image));
/* Return value is one of: */
#define JPEG_SUSPENDED 0 /* Suspended due to lack of input data */
#define JPEG_HEADER_OK 1 /* Found valid image datastream */
#define JPEG_HEADER_TABLES_ONLY 2 /* Found valid table-specs-only datastream */
/* If you pass require_image = TRUE (normal case), you need not check for
 * a TABLES_ONLY return code; an abbreviated file will cause an error exit.
 * JPEG_SUSPENDED is only possible if you use a data source module that can
 * give a suspension return (the stdio source module doesn't).
 */

/* Main entry points for decompression */
EXTERN(boolean) jpeg_start_decompress JPP((j_decompress_ptr cinfo));
EXTERN(JDIMENSION) jpeg_read_scanlines JPP((j_decompress_ptr cinfo,
        JSAMPARRAY scanlines,
        JDIMENSION max_lines));
EXTERN(boolean) jpeg_finish_decompress JPP((j_decompress_ptr cinfo));

/* Replaces jpeg_read_scanlines when reading raw downsampled data. */
EXTERN(JDIMENSION) jpeg_read_raw_data JPP((j_decompress_ptr cinfo,
        JSAMPIMAGE data,
        JDIMENSION max_lines));

/* Additional entry points for buffered-image mode. */
EXTERN(boolean) jpeg_has_multiple_scans JPP((j_decompress_ptr cinfo));
EXTERN(boolean) jpeg_start_output JPP((j_decompress_ptr cinfo,
        int scan_number));
EXTERN(boolean) jpeg_finish_output JPP((j_decompress_ptr cinfo));
EXTERN(boolean) jpeg_input_complete JPP((j_decompress_ptr cinfo));
EXTERN(void) jpeg_new_colormap JPP((j_decompress_ptr cinfo));
EXTERN(int) jpeg_consume_input JPP((j_decompress_ptr cinfo));
/* Return value is one of: */
#define JPEG_SUSPENDED 0 /* Suspended due to lack of input data */
#define JPEG_REACHED_SOS 1 /* Reached start of new scan */
#define JPEG_REACHED_EOI 2 /* Reached end of image */
#define JPEG_ROW_COMPLETED 3 /* Completed one iMCU row */
#define JPEG_SCAN_COMPLETED 4 /* Completed last iMCU row of a scan */

/* Precalculate output dimensions for current decompression parameters. */
EXTERN(void) jpeg_calc_output_dimensions JPP((j_decompress_ptr cinfo));

```

```

/* Control saving of COM and APPn markers into marker_list. */
EXTERN(void) jpeg_save_markers
    JPP((j_decompress_ptr cinfo, int marker_code,
        unsigned int length_limit));

/* Install a special processing method for COM or APPn markers. */
EXTERN(void) jpeg_set_marker_processor
    JPP((j_decompress_ptr cinfo, int marker_code,
        jpeg_marker_parser_method routine));

/* Read or write raw DCT coefficients --- useful for lossless transcoding. */
EXTERN(jvirt_barray_ptr *) jpeg_read_coefficients JPP((j_decompress_ptr cinfo));
EXTERN(void) jpeg_write_coefficients JPP((j_compress_ptr cinfo,
    jvirt_barray_ptr * coef_arrays));
EXTERN(void) jpeg_copy_critical_parameters JPP((j_decompress_ptr srcinfo,
    j_compress_ptr dstinfo));

/* If you choose to abort compression or decompression before completing
 * jpeg_finish_(de)compress, then you need to clean up to release memory,
 * temporary files, etc. You can just call jpeg_destroy_(de)compress
 * if you're done with the JPEG object, but if you want to clean it up and
 * reuse it, call this:
 */
EXTERN(void) jpeg_abort_compress JPP((j_compress_ptr cinfo));
EXTERN(void) jpeg_abort_decompress JPP((j_decompress_ptr cinfo));

/* Generic versions of jpeg_abort and jpeg_destroy that work on either
 * flavor of JPEG object. These may be more convenient in some places.
 */
EXTERN(void) jpeg_abort JPP((j_common_ptr cinfo));
EXTERN(void) jpeg_destroy JPP((j_common_ptr cinfo));

/* Default restart-marker-resync procedure for use by data source modules */
EXTERN(boolean) jpeg_resync_to_restart JPP((j_decompress_ptr cinfo,
    int desired));

/* These marker codes are exported since applications and data source modules
 * are likely to want to use them.
 */
#define JPEG_RST0 0xD0 /* RST0 marker code */
#define JPEG_EOI 0xD9 /* EOI marker code */
#define JPEG_APP0 0xE0 /* APP0 marker code */
#define JPEG_COM 0xFE /* COM marker code */

/* If we have a brain-damaged compiler that emits warnings (or worse, errors)
 * for structure definitions that are never filled in, keep it quiet by
 * supplying dummy definitions for the various substructures.
 */
#ifdef INCOMPLETE_TYPES_BROKEN
#ifndef JPEG_INTERNALS /* will be defined in jpegint.h */
struct jvirt_sarray_control { long dummy; };
struct jvirt_barray_control { long dummy; };
struct jpeg_comp_master { long dummy; };
struct jpeg_c_main_controller { long dummy; };
struct jpeg_c_prep_controller { long dummy; };
struct jpeg_c_coef_controller { long dummy; };
struct jpeg_marker_writer { long dummy; };
struct jpeg_color_converter { long dummy; };
struct jpeg_downsampler { long dummy; };
struct jpeg_forward_dct { long dummy; };
struct jpeg_entropy_encoder { long dummy; };
struct jpeg_decomp_master { long dummy; };
struct jpeg_d_main_controller { long dummy; };
struct jpeg_d_coef_controller { long dummy; };
struct jpeg_d_post_controller { long dummy; };
struct jpeg_input_controller { long dummy; };
struct jpeg_marker_reader { long dummy; };
struct jpeg_entropy_decoder { long dummy; };
struct jpeg_inverse_dct { long dummy; };
struct jpeg_upsampler { long dummy; };
struct jpeg_color_deconverter { long dummy; };
struct jpeg_color_quantizer { long dummy; };
#endif /* JPEG_INTERNALS */
#endif /* INCOMPLETE_TYPES_BROKEN */

```

```

/*
 * The JPEG library modules define JPEG_INTERNALS before including this file.
 * The internal structure declarations are read only when that is true.
 * Applications using the library should not include jpegint.h, but may wish
 * to include jerror.h.
 */

#ifdef JPEG_INTERNALS
#include "jpegint.h"          /* fetch private declarations */
#include "jerror.h"          /* fetch error codes too */
#endif

#endif /* JPEGLIB_H */

```



```

/*
 * jversion.h
 *
 * Copyright (C) 1991-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains software version identification.
 */

```

```
#define JVERSION      "6b  27-Mar-1998"
```

```
#define JCOPYRIGHT "Copyright (C) 1998, Thomas G. Lane"
```

```

////////////////////////////////////
// Construction/Destruction
////////////////////////////////////

RawPixelEncoder::RawPixelEncoder()
{
    m_ReleaseBufferMemory=false;
    m_pBuffer=NULL;
    m_BufferPtr=0;
}

RawPixelEncoder::~RawPixelEncoder() { ReleaseBuffer(); }

/*****
*
*   Prepare buffer for data entry
*
*****/
bool RawPixelEncoder::SetSize(UINT32 size)
{
    ReleaseBuffer();
    if(size<=0) return false;
    m_Size=size;
    try
    {
        m_pBuffer = new BYTE[m_Size];
    }
    catch(...) { return false; }
    if(m_pBuffer)
    {
        memset(m_pBuffer,0,m_Size);
        m_ReleaseBufferMemory=true;
        m_BufferPtr=0;
        return true;
    }
    else return false;
}

/*****
*
*   Add new subbufer
*
*****/
UINT32 RawPixelEncoder::AddData(BYTE* buf, UINT32 buf_size,
                                UINT32 fliprawbytes /*=0*/)
{
    bool flipY = (fliprawbytes>1);
    if(m_BufferPtr>=m_Size) return 0;
    if(buf_size>m_Size-m_BufferPtr) // avoid overflow
    {
        flipY=false;
        buf_size = m_Size-m_BufferPtr;
    }
    BYTE *start = &(m_pBuffer[m_BufferPtr]);
    for(UINT32 k=0; k<buf_size; k++)
    {
        m_pBuffer[m_BufferPtr] = buf[k];
        m_BufferPtr++;
    }
    if(flipY)
    {
        ::Flip2DBufferY(start, buf_size, fliprawbytes);
    }
    start=NULL;
    return buf_size;
}

/*****
*
*   Transfer buffer data to VR
*
*****/
void RawPixelEncoder::TransferDataToVR(VR *vr)
{
    vr->AttachData(m_pBuffer,m_Size);
    m_pBuffer=NULL;
    m_ReleaseBufferMemory=FALSE;
}

/*****
*

```

\* Release allocated buffer memory

\*

\*\*\*\*\*/

void RawPixelEncoder::ReleaseBuffer()

```
{
    if(m_ReleaseBufferMemory)
    {
        if(m_pBuffer)
        {
            delete [] m_pBuffer;
            m_pBuffer=NULL;
        }
        m_ReleaseBufferMemory=false;
    }
}
```

```

/*
 * jchuff.h
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains declarations for Huffman entropy encoding routines
 * that are shared between the sequential encoder (jchuff.c) and the
 * progressive encoder (jcphuff.c). No other modules need to see these.
 */

/* The legal range of a DCT coefficient is
 * -1024 .. +1023 for 8-bit data;
 * -16384 .. +16383 for 12-bit data.
 * Hence the magnitude should always fit in 10 or 14 bits respectively.
 */

#if BITS_IN_JSAMPLE == 8
#define MAX_COEF_BITS 10
#else
#define MAX_COEF_BITS 14
#endif

/* Derived data constructed for each Huffman table */
typedef struct {
  unsigned int ehufco[256]; /* code for each symbol */
  char ehufsi[256]; /* length of code for each symbol */
  /* If no code has been allocated for a symbol S, ehufsi[S] contains 0 */
} c_derived_tbl;

/* Short forms of external names for systems with brain-damaged linkers. */
#ifdef NEED_SHORT_EXTERNAL_NAMES
#define jpeg_make_c_derived_tbl jMkCDerived
#define jpeg_gen_optimal_table jGenOptTbl
#endif /* NEED_SHORT_EXTERNAL_NAMES */

/* Expand a Huffman table definition into the derived format */
EXTERN(void) jpeg_make_c_derived_tbl
  JPP((j_compress_ptr cinfo, boolean isDC, int tblno,
       c_derived_tbl ** pdtbl));

/* Generate an optimal table definition given the specified counts */
EXTERN(void) jpeg_gen_optimal_table
  JPP((j_compress_ptr cinfo, JHUFF_TBL * htbl, long freq[]));

```

```

/* jconfig.vc --- jconfig.h for Microsoft Visual C++ on Windows 95 or NT. */
/* see jconfig.doc for explanations */

```

```

#define HAVE_PROTOTYPES
#define HAVE_UNSIGNED_CHAR
#define HAVE_UNSIGNED_SHORT
/* #define void char */
/* #define const */
#undef CHAR_IS_UNSIGNED
#define HAVE_STDDEF_H
#define HAVE_STDLIB_H
#undef NEED_BSD_STRINGS
#undef NEED_SYS_TYPES_H
#undef NEED_FAR_POINTERS /* we presume a 32-bit flat memory model */
#undef NEED_SHORT_EXTERNAL_NAMES
#undef INCOMPLETE_TYPES_BROKEN

```

```

/* Define "boolean" as unsigned char, not int, per Windows custom */
#ifndef __RPCNDR_H /* don't conflict if rpcnldr.h already read */
typedef unsigned char boolean;
#endif
#define HAVE_BOOLEAN /* prevent jmorecfg.h from redefining it */

```

```

#ifdef JPEG_INTERNALS
#undef RIGHT_SHIFT_IS_UNSIGNED
#endif /* JPEG_INTERNALS */

```

```

#ifdef JPEG_CJPEG_DJPEG
#define BMP_SUPPORTED /* BMP image file format */
#define GIF_SUPPORTED /* GIF image file format */
#define PPM_SUPPORTED /* PBMPLUS PPM/PGM image file format */
#undef RLE_SUPPORTED /* Utah RLE image file format */
#define TARGA_SUPPORTED /* Targa image file format */

#define TWO_FILE_COMMANDLINE /* optional */
#define USE_SETMODE /* Microsoft has setmode() */
#undef NEED_SIGNAL_CATCHER
#undef DONT_USE_B_MODE
#undef PROGRESS_REPORT /* optional */
#endif /* JPEG_CJPEG_DJPEG */

```

```

/*
 * jdct.h
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This include file contains common declarations for the forward and
 * inverse DCT modules. These declarations are private to the DCT managers
 * (jcdctmgr.c, jddctmgr.c) and the individual DCT algorithms.
 * The individual DCT algorithms are kept in separate files to ease
 * machine-dependent tuning (e.g., assembly coding).
 */

/*
 * A forward DCT routine is given a pointer to a work area of type DCTELEM[];
 * the DCT is to be performed in-place in that buffer. Type DCTELEM is int
 * for 8-bit samples, INT32 for 12-bit samples. (NOTE: Floating-point DCT
 * implementations use an array of type FAST_FLOAT, instead.)
 * The DCT inputs are expected to be signed (range +-CENTERJSAMPLE).
 * The DCT outputs are returned scaled up by a factor of 8; they therefore
 * have a range of +-8K for 8-bit data, +-128K for 12-bit data. This
 * convention improves accuracy in integer implementations and saves some
 * work in floating-point ones.
 * Quantization of the output coefficients is done by jcdctmgr.c.
 */

#if BITS_IN_JSAMPLE == 8
typedef int DCTELEM; /* 16 or 32 bits is fine */
#else
typedef INT32 DCTELEM; /* must have 32 bits */
#endif

typedef JMETHOD(void, forward_DCT_method_ptr, (DCTELEM * data));
typedef JMETHOD(void, float_DCT_method_ptr, (FAST_FLOAT * data));

/*
 * An inverse DCT routine is given a pointer to the input JBLOCK and a pointer
 * to an output sample array. The routine must dequantize the input data as
 * well as perform the IDCT; for dequantization, it uses the multiplier table
 * pointed to by compptr-> dct_table. The output data is to be placed into the
 * sample array starting at a specified column. (Any row offset needed will
 * be applied to the array pointer before it is passed to the IDCT code.)
 * Note that the number of samples emitted by the IDCT routine is
 * DCT_scaled_size * DCT_scaled_size.
 */

/* typedef inverse_DCT_method_ptr is declared in jpegint.h */

/* Each IDCT routine has its own ideas about the best dct_table element type.
 */

typedef MULTIPLIER ISLOW_MULT_TYPE; /* short or int, whichever is faster */
#if BITS_IN_JSAMPLE == 8
typedef MULTIPLIER IFAST_MULT_TYPE; /* 16 bits is OK, use short if faster */
#define IFAST_SCALE_BITS 2 /* fractional bits in scale factors */
#else
typedef INT32 IFAST_MULT_TYPE; /* need 32 bits for scaled quantizers */
#define IFAST_SCALE_BITS 13 /* fractional bits in scale factors */
#endif
typedef FAST_FLOAT FLOAT_MULT_TYPE; /* preferred floating type */

/*
 * Each IDCT routine is responsible for range-limiting its results and
 * converting them to unsigned form (0..MAXJSAMPLE). The raw outputs could
 * be quite far out of range if the input data is corrupt, so a bulletproof
 * range-limiting step is required. We use a mask-and-table-lookup method
 * to do the combined operations quickly. See the comments with
 * prepare_range_limit_table (in jdmaster.c) for more info.
 */

#define IDCT_range_limit(cinfo) ((cinfo)->sample_range_limit + CENTERJSAMPLE)
#define RANGE_MASK (MAXJSAMPLE * 4 + 3) /* 2 bits wider than legal samples */

```

```
/* Short forms of external names for systems with brain-damaged linkers. */
```

```
#ifdef NEED_SHORT_EXTERNAL_NAMES
#define jpeg_fdct_islow    jFDIslow
#define jpeg_fdct_ifast    jFDIfast
#define jpeg_fdct_float    jFDfloat
#define jpeg_idct_islow    jRDIslow
#define jpeg_idct_ifast    jRDIfast
#define jpeg_idct_float    jRDfloat
#define jpeg_idct_4x4      jRD4x4
#define jpeg_idct_2x2      jRD2x2
#define jpeg_idct_1x1      jRD1x1
#endif /* NEED_SHORT_EXTERNAL_NAMES */
```

```
/* Extern declarations for the forward and inverse DCT routines. */
```

```
EXTERN(void) jpeg_fdct_islow JPP((DCTELEM * data));
EXTERN(void) jpeg_fdct_ifast JPP((DCTELEM * data));
EXTERN(void) jpeg_fdct_float JPP((FAST_FLOAT * data));

EXTERN(void) jpeg_idct_islow
    JPP((j_decompress_ptr cinfo, jpeg_component_info * compptr,
        JOEFPTR coef_block, JSAMPARRAY output_buf, JDIMENSION output_col));
EXTERN(void) jpeg_idct_ifast
    JPP((j_decompress_ptr cinfo, jpeg_component_info * compptr,
        JOEFPTR coef_block, JSAMPARRAY output_buf, JDIMENSION output_col));
EXTERN(void) jpeg_idct_float
    JPP((j_decompress_ptr cinfo, jpeg_component_info * compptr,
        JOEFPTR coef_block, JSAMPARRAY output_buf, JDIMENSION output_col));
EXTERN(void) jpeg_idct_4x4
    JPP((j_decompress_ptr cinfo, jpeg_component_info * compptr,
        JOEFPTR coef_block, JSAMPARRAY output_buf, JDIMENSION output_col));
EXTERN(void) jpeg_idct_2x2
    JPP((j_decompress_ptr cinfo, jpeg_component_info * compptr,
        JOEFPTR coef_block, JSAMPARRAY output_buf, JDIMENSION output_col));
EXTERN(void) jpeg_idct_1x1
    JPP((j_decompress_ptr cinfo, jpeg_component_info * compptr,
        JOEFPTR coef_block, JSAMPARRAY output_buf, JDIMENSION output_col));
```

```
/*
 * Macros for handling fixed-point arithmetic; these are used by many
 * but not all of the DCT/IDCT modules.
 *
 * All values are expected to be of type INT32.
 * Fractional constants are scaled left by CONST_BITS bits.
 * CONST_BITS is defined within each module using these macros,
 * and may differ from one module to the next.
 */
```

```
#define ONE ((INT32) 1)
#define CONST_SCALE (ONE << CONST_BITS)
```

```
/* Convert a positive real constant to an integer scaled by CONST_SCALE.
 * Caution: some C compilers fail to reduce "FIX(constant)" at compile time,
 * thus causing a lot of useless floating-point operations at run time.
 */
```

```
#define FIX(x) ((INT32) ((x) * CONST_SCALE + 0.5))
```

```
/* Descale and correctly round an INT32 value that's scaled by N bits.
 * We assume RIGHT_SHIFT rounds towards minus infinity, so adding
 * the fudge factor is correct for either sign of X.
 */
```

```
#define DESCALE(x,n) RIGHT_SHIFT((x) + (ONE << ((n)-1)), n)
```

```
/* Multiply an INT32 variable by an INT32 constant to yield an INT32 result.
 * This macro is used only when the two inputs will actually be no more than
 * 16 bits wide, so that a 16x16->32 bit multiply can be used instead of a
 * full 32x32 multiply. This provides a useful speedup on many machines.
 * Unfortunately there is no way to specify a 16x16->32 multiply portably
 * in C, but some C compilers will do the right thing if you provide the
 * correct combination of casts.
 */
```

```
#ifdef SHORTxSHORT_32 /* may work if 'int' is 32 bits */
#define MULTIPLY16C16(var,const) (((INT16) (var)) * ((INT16) (const)))
#endif
#ifdef SHORTxLCONST_32 /* known to work with Microsoft C 6.0 */
```

```

#define MULTIPLY16C16(var,const) (((INT16) (var)) * ((INT32) (const)))
#endif

#ifdef MULTIPLY16C16 /* default definition */
#define MULTIPLY16C16(var,const) ((var) * (const))
#endif

/* Same except both inputs are variables. */

#ifdef SHORTxSHORT_32 /* may work if 'int' is 32 bits */
#define MULTIPLY16V16(var1,var2) (((INT16) (var1)) * ((INT16) (var2)))
#endif

#ifdef MULTIPLY16V16 /* default definition */
#define MULTIPLY16V16(var1,var2) ((var1) * (var2))
#endif

```



```

/*
 * jdhuff.h
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains declarations for Huffman entropy decoding routines
 * that are shared between the sequential decoder (jdhuff.c) and the
 * progressive decoder (jdpuff.c).  No other modules need to see these.
 */

/* Short forms of external names for systems with brain-damaged linkers. */

#ifdef NEED_SHORT_EXTERNAL_NAMES
#define jpeg_make_d_derived_tbl  jMkDDerived
#define jpeg_fill_bit_buffer      jFilBitBuf
#define jpeg_huff_decode          jHufDecode
#endif /* NEED_SHORT_EXTERNAL_NAMES */

/* Derived data constructed for each Huffman table */

#define HUFF_LOOKAHEAD 8 /* # of bits of lookahead */

typedef struct {
  /* Basic tables: (element [0] of each array is unused) */
  INT32 maxcode[18]; /* largest code of length k (-1 if none) */
  /* (maxcode[17] is a sentinel to ensure jpeg_huff_decode terminates) */
  INT32 valoffset[17]; /* huffval[] offset for codes of length k */
  /* valoffset[k] = huffval[] index of 1st symbol of code length k, less
   * the smallest code of length k; so given a code of length k, the
   * corresponding symbol is huffval[code + valoffset[k]]
   */
  /* Link to public Huffman table (needed only in jpeg_huff_decode) */
  JHUFF_TBL *pub;

  /* Lookahead tables: indexed by the next HUFF_LOOKAHEAD bits of
   * the input data stream.  If the next Huffman code is no more
   * than HUFF_LOOKAHEAD bits long, we can obtain its length and
   * the corresponding symbol directly from these tables.
   */
  int look_nbits[1<<HUFF_LOOKAHEAD]; /* # bits, or 0 if too long */
  UINT8 look_sym[1<<HUFF_LOOKAHEAD]; /* symbol, or unused */
} d_derived_tbl;

/* Expand a Huffman table definition into the derived format */
EXTERN(void) jpeg_make_d_derived_tbl
  JPP((j_decompress_ptr cinfo, boolean isDC, int tblno,
       d_derived_tbl ** pdtbl));

/*
 * Fetching the next N bits from the input stream is a time-critical operation
 * for the Huffman decoders.  We implement it with a combination of inline
 * macros and out-of-line subroutines.  Note that N (the number of bits
 * demanded at one time) never exceeds 15 for JPEG use.
 *
 * We read source bytes into get_buffer and dole out bits as needed.
 * If get_buffer already contains enough bits, they are fetched in-line
 * by the macros CHECK_BIT_BUFFER and GET_BITS.  When there aren't enough
 * bits, jpeg_fill_bit_buffer is called; it will attempt to fill get_buffer
 * as full as possible (not just to the number of bits needed; this
 * prefetching reduces the overhead cost of calling jpeg_fill_bit_buffer).
 * Note that jpeg_fill_bit_buffer may return FALSE to indicate suspension.
 * On TRUE return, jpeg_fill_bit_buffer guarantees that get_buffer contains
 * at least the requested number of bits --- dummy zeroes are inserted if
 * necessary.
 */

typedef INT32 bit_buf_type; /* type of bit-extraction buffer */
#define BIT_BUF_SIZE 32 /* size of buffer in bits */

/* If long is > 32 bits on your machine, and shifting/masking longs is
 * reasonably fast, making bit_buf_type be long and setting BIT_BUF_SIZE
 * appropriately should be a win.  Unfortunately we can't define the size
 * with something like #define BIT_BUF_SIZE (sizeof(bit_buf_type)*8)
 * because not all machines measure sizeof in 8-bit bytes.
 */

```

```

typedef struct {          /* Bitreading state saved across MCUs */
    bit_buf_type get_buffer; /* current bit-extraction buffer */
    int bits_left;          /* # of unused bits in it */
} bitread_perm_state;

typedef struct {          /* Bitreading working state within an MCU */
    /* Current data source location */
    /* We need a copy, rather than munging the original, in case of suspension */
    const JOCTET * next_input_byte; /* => next byte to read from source */
    size_t bytes_in_buffer; /* # of bytes remaining in source buffer */
    /* Bit input buffer --- note these values are kept in register variables,
     * not in this struct, inside the inner loops.
     */
    bit_buf_type get_buffer; /* current bit-extraction buffer */
    int bits_left;          /* # of unused bits in it */
    /* Pointer needed by jpeg_fill_bit_buffer. */
    j_decompress_ptr cinfo; /* back link to decompress master record */
} bitread_working_state;

/* Macros to declare and load/save bitread local variables. */
#define BITREAD_STATE_VARS \
    register bit_buf_type get_buffer; \
    register int bits_left; \
    bitread_working_state br_state

#define BITREAD_LOAD_STATE(cinfo,permstate) \
    br_state.cinfo = cinfo; \
    br_state.next_input_byte = cinfo->src->next_input_byte; \
    br_state.bytes_in_buffer = cinfo->src->bytes_in_buffer; \
    get_buffer = permstate.get_buffer; \
    bits_left = permstate.bits_left;

#define BITREAD_SAVE_STATE(cinfo,permstate) \
    cinfo->src->next_input_byte = br_state.next_input_byte; \
    cinfo->src->bytes_in_buffer = br_state.bytes_in_buffer; \
    permstate.get_buffer = get_buffer; \
    permstate.bits_left = bits_left

/*
 * These macros provide the in-line portion of bit fetching.
 * Use CHECK_BIT_BUFFER to ensure there are N bits in get_buffer
 * before using GET_BITS, PEEK_BITS, or DROP_BITS.
 * The variables get_buffer and bits_left are assumed to be locals,
 * but the state struct might not be (jpeg_huff_decode needs this).
 * CHECK_BIT_BUFFER(state,n,action);
 *   Ensure there are N bits in get_buffer; if suspend, take action.
 *   val = GET_BITS(n);
 *   Fetch next N bits.
 *   val = PEEK_BITS(n);
 *   Fetch next N bits without removing them from the buffer.
 * DROP_BITS(n);
 *   Discard next N bits.
 * The value N should be a simple variable, not an expression, because it
 * is evaluated multiple times.
 */

#define CHECK_BIT_BUFFER(state,nbits,action) \
    { if (bits_left < (nbits)) { \
        if (! jpeg_fill_bit_buffer(&(state),get_buffer,bits_left,nbits)) \
            { action; } \
        get_buffer = (state).get_buffer; bits_left = (state).bits_left; } }

#define GET_BITS(nbits) \
    (((int) (get_buffer >> (bits_left -= (nbits)))) & ((1<<(nbits))-1))

#define PEEK_BITS(nbits) \
    (((int) (get_buffer >> (bits_left - (nbits)))) & ((1<<(nbits))-1))

#define DROP_BITS(nbits) \
    (bits_left -= (nbits))

/* Load up the bit buffer to a depth of at least nbits */
EXTERN(boolean) jpeg_fill_bit_buffer
JPP((bitread_working_state * state, register bit_buf_type get_buffer,
    register int bits_left, int nbits));

/*
 * Code for extracting next Huffman-coded symbol from input bit stream.

```

\* Again, this is time-critical and we make the main paths be macros.  
 \* We use a lookahead table to process codes of up to HUFF\_LOOKAHEAD bits  
 \* without looping. Usually, more than 95% of the Huffman codes will be 8  
 \* or fewer bits long. The few overlength codes are handled with a loop,  
 \* which need not be inline code.

\* Notes about the HUFF\_DECODE macro:

- \* 1. Near the end of the data segment, we may fail to get enough bits  
 \* for a lookahead. In that case, we do it the hard way.
- \* 2. If the lookahead table contains no entry, the next code must be  
 \* more than HUFF\_LOOKAHEAD bits long.
- \* 3. jpeg\_huff\_decode returns -1 if forced to suspend.

\*/

```
#define HUFF_DECODE(result,state,htbl,failaction,slowlabel) \
{ register int nb, look; \
  if (bits_left < HUFF_LOOKAHEAD) { \
    if (! jpeg_fill_bit_buffer(&state,get_buffer,bits_left, 0)) {failaction;} \
    get_buffer = state.get_buffer; bits_left = state.bits_left; \
    if (bits_left < HUFF_LOOKAHEAD) { \
      nb = 1; goto slowlabel; \
    } \
  } \
  look = PEEK_BITS(HUFF_LOOKAHEAD); \
  if ((nb = htbl->look_nbits[look]) != 0) { \
    DROP_BITS(nb); \
    result = htbl->look_sym[look]; \
  } else { \
    nb = HUFF_LOOKAHEAD+1; \
slowlabel: \
    if ((result=jpeg_huff_decode(&state,get_buffer,bits_left,htbl,nb)) < 0) \
    { failaction; } \
    get_buffer = state.get_buffer; bits_left = state.bits_left; \
  } \
}
```

/\* Out-of-line case for Huffman code fetching \*/

```
EXTERN(int) jpeg_huff_decode
JPP((bitread_working_state * state, register bit_buf_type get_buffer,
     register int bits_left, d_derived_tbl * htbl, int min_bits));
```

```

/*
 * jerror.h
 *
 * Copyright (C) 1994-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file defines the error and message codes for the JPEG library.
 * Edit this file to add new codes, or to translate the message strings to
 * some other language.
 * A set of error-reporting macros are defined too. Some applications using
 * the JPEG library may wish to include this file to get the error codes
 * and/or the macros.
 */

/*
 * To define the enum list of message codes, include this file without
 * defining macro JMESSAGE. To create a message string table, include it
 * again with a suitable JMESSAGE definition (see jerror.c for an example).
 */
#ifndef JMESSAGE
#ifndef JERROR_H
/* First time through, define the enum list */
#define JMAKE_ENUM_LIST
#else
/* Repeated inclusions of this file are no-ops unless JMESSAGE is defined */
#define JMESSAGE(code,string)
#endif /* JERROR_H */
#endif /* JMESSAGE */

#ifdef JMAKE_ENUM_LIST

typedef enum {
#define JMESSAGE(code,string) code,
#define JMESSAGE(JMSG_NOMESSAGE, "Bogus message code %d") /* Must be first entry! */

/* For maintenance convenience, list is alphabetical by message code name */
JMESSAGE(JERR_ARITH_NOTIMPL,
  "Sorry, there are legal restrictions on arithmetic coding")
JMESSAGE(JERR_BAD_ALIGN_TYPE, "ALIGN_TYPE is wrong, please fix")
JMESSAGE(JERR_BAD_ALLOC_CHUNK, "MAX_ALLOC_CHUNK is wrong, please fix")
JMESSAGE(JERR_BAD_BUFFER_MODE, "Bogus buffer control mode")
JMESSAGE(JERR_BAD_COMPONENT_ID, "Invalid component ID %d in SOS")
JMESSAGE(JERR_BAD_DCT_COEF, "DCT coefficient out of range")
JMESSAGE(JERR_BAD_DCTSIZE, "IDCT output block size %d not supported")
JMESSAGE(JERR_BAD_HUFF_TABLE, "Bogus Huffman table definition")
JMESSAGE(JERR_BAD_IN_COLORSPACE, "Bogus input colorspace")
JMESSAGE(JERR_BAD_J_COLORSPACE, "Bogus JPEG colorspace")
JMESSAGE(JERR_BAD_LENGTH, "Bogus marker length")
JMESSAGE(JERR_BAD_LIB_VERSION,
  "Wrong JPEG library version: library is %d, caller expects %d")
JMESSAGE(JERR_BAD_MCU_SIZE, "Sampling factors too large for interleaved scan")
JMESSAGE(JERR_BAD_POOL_ID, "Invalid memory pool code %d")
JMESSAGE(JERR_BAD_PRECISION, "Unsupported JPEG data precision %d")
JMESSAGE(JERR_BAD_PROGRESSION,
  "Invalid progressive parameters Ss=%d Se=%d Ah=%d Al=%d")
JMESSAGE(JERR_BAD_PROG_SCRIPT,
  "Invalid progressive parameters at scan script entry %d")
JMESSAGE(JERR_BAD_SAMPLING, "Bogus sampling factors")
JMESSAGE(JERR_BAD_SCAN_SCRIPT, "Invalid scan script at entry %d")
JMESSAGE(JERR_BAD_STATE, "Improper call to JPEG library in state %d")
JMESSAGE(JERR_BAD_STRUCT_SIZE,
  "JPEG parameter struct mismatch: library thinks size is %u, caller expects %u")
JMESSAGE(JERR_BAD_VIRTUAL_ACCESS, "Bogus virtual array access")
JMESSAGE(JERR_BUFFER_SIZE, "Buffer passed to JPEG library is too small")
JMESSAGE(JERR_CANT_SUSPEND, "Suspension not allowed here")
JMESSAGE(JERR_CCIR601_NOTIMPL, "CCIR601 sampling not implemented yet")
JMESSAGE(JERR_COMPONENT_COUNT, "Too many color components: %d, max %d")
JMESSAGE(JERR_CONVERSION_NOTIMPL, "Unsupported color conversion request")
JMESSAGE(JERR_DAC_INDEX, "Bogus DAC index %d")
JMESSAGE(JERR_DAC_VALUE, "Bogus DAC value 0x%x")
JMESSAGE(JERR_DHT_INDEX, "Bogus DHT index %d")
JMESSAGE(JERR_DQT_INDEX, "Bogus DQT index %d")
JMESSAGE(JERR_EMPTY_IMAGE, "Empty JPEG image (DNL not supported)")
JMESSAGE(JERR_EMS_READ, "Read from EMS failed")
JMESSAGE(JERR_EMS_WRITE, "Write to EMS failed")

```

```

JMESSAGE(JERR_EOI_EXPECTED, "Didn't expect more than one scan")
JMESSAGE(JERR_FILE_READ, "Input file read error")
JMESSAGE(JERR_FILE_WRITE, "Output file write error --- out of disk space?")
JMESSAGE(JERR_FRACT_SAMPLE_NOTIMPL, "Fractional sampling not implemented yet")
JMESSAGE(JERR_HUFF_CLEN_OVERFLOW, "Huffman code size table overflow")
JMESSAGE(JERR_HUFF_MISSING_CODE, "Missing Huffman code table entry")
JMESSAGE(JERR_IMAGE_TOO_BIG, "Maximum supported image dimension is %u pixels")
JMESSAGE(JERR_INPUT_EMPTY, "Empty input file")
JMESSAGE(JERR_INPUT_EOF, "Premature end of input file")
JMESSAGE(JERR_MISMATCHED_QUANT_TABLE,
    "Cannot transcode due to multiple use of quantization table %d")
JMESSAGE(JERR_MISSING_DATA, "Scan script does not transmit all data")
JMESSAGE(JERR_MODE_CHANGE, "Invalid color quantization mode change")
JMESSAGE(JERR_NOTIMPL, "Not implemented yet")
JMESSAGE(JERR_NOT_COMPILED, "Requested feature was omitted at compile time")
JMESSAGE(JERR_NO_BACKING_STORE, "Backing store not supported")
JMESSAGE(JERR_NO_HUFF_TABLE, "Huffman table 0x%02x was not defined")
JMESSAGE(JERR_NO_IMAGE, "JPEG datastream contains no image")
JMESSAGE(JERR_NO_QUANT_TABLE, "Quantization table 0x%02x was not defined")
JMESSAGE(JERR_NO_SOI, "Not a JPEG file: starts with 0x%02x 0x%02x")
JMESSAGE(JERR_OUT_OF_MEMORY, "Insufficient memory (case %d)")
JMESSAGE(JERR_QUANT_COMPONENTS,
    "Cannot quantize more than %d color components")
JMESSAGE(JERR_QUANT_FEW_COLORS, "Cannot quantize to fewer than %d colors")
JMESSAGE(JERR_QUANT_MANY_COLORS, "Cannot quantize to more than %d colors")
JMESSAGE(JERR_SOF_DUPLICATE, "Invalid JPEG file structure: two SOF markers")
JMESSAGE(JERR_SOF_NO_SOS, "Invalid JPEG file structure: missing SOS marker")
JMESSAGE(JERR_SOF_UNSUPPORTED, "Unsupported JPEG process: SOF type 0x%02x")
JMESSAGE(JERR_SOI_DUPLICATE, "Invalid JPEG file structure: two SOI markers")
JMESSAGE(JERR_SOS_NO_SOF, "Invalid JPEG file structure: SOS before SOF")
JMESSAGE(JERR_TFILE_CREATE, "Failed to create temporary file %s")
JMESSAGE(JERR_TFILE_READ, "Read failed on temporary file")
JMESSAGE(JERR_TFILE_SEEK, "Seek failed on temporary file")
JMESSAGE(JERR_TFILE_WRITE,
    "Write failed on temporary file --- out of disk space?")
JMESSAGE(JERR_TOO_LITTLE_DATA, "Application transferred too few scanlines")
JMESSAGE(JERR_UNKNOWN_MARKER, "Unsupported marker type 0x%02x")
JMESSAGE(JERR_VIRTUAL_BUG, "Virtual array controller messed up")
JMESSAGE(JERR_WIDTH_OVERFLOW, "Image too wide for this implementation")
JMESSAGE(JERR_XMS_READ, "Read from XMS failed")
JMESSAGE(JERR_XMS_WRITE, "Write to XMS failed")
JMESSAGE(JMSG_COPYRIGHT, JCOPYRIGHT)
JMESSAGE(JMSG_VERSION, JVERSION)
JMESSAGE(JTRC_16BIT_TABLES,
    "Caution: quantization tables are too coarse for baseline JPEG")
JMESSAGE(JTRC_ADOBE,
    "Adobe APP14 marker: version %d, flags 0x%04x 0x%04x, transform %d")
JMESSAGE(JTRC_APP0, "Unknown APP0 marker (not JFIF), length %u")
JMESSAGE(JTRC_APP14, "Unknown APP14 marker (not Adobe), length %u")
JMESSAGE(JTRC_DAC, "Define Arithmetic Table 0x%02x: 0x%02x")
JMESSAGE(JTRC_DHT, "Define Huffman Table 0x%02x")
JMESSAGE(JTRC_DQT, "Define Quantization Table %d precision %d")
JMESSAGE(JTRC_DRI, "Define Restart Interval %u")
JMESSAGE(JTRC_EMS_CLOSE, "Freed EMS handle %u")
JMESSAGE(JTRC_EMS_OPEN, "Obtained EMS handle %u")
JMESSAGE(JTRC_EOI, "End Of Image")
JMESSAGE(JTRC_HUFFBITS, "    %3d %3d %3d %3d %3d %3d %3d %3d")
JMESSAGE(JTRC_JFIF, "JFIF APP0 marker: version %d.%02d, density %dx%d %d")
JMESSAGE(JTRC_JFIF_BADTHUMBNAILSIZE,
    "Warning: thumbnail image size does not match data length %u")
JMESSAGE(JTRC_JFIF_EXTENSION,
    "JFIF extension marker: type 0x%02x, length %u")
JMESSAGE(JTRC_JFIF_THUMBNAIL, "    with %d x %d thumbnail image")
JMESSAGE(JTRC_MISC_MARKER, "Miscellaneous marker 0x%02x, length %u")
JMESSAGE(JTRC_PARMLESS_MARKER, "Unexpected marker 0x%02x")
JMESSAGE(JTRC_QUANTVALS, "    %4u %4u %4u %4u %4u %4u %4u %4u")
JMESSAGE(JTRC_QUANT_3_COLORS, "Quantizing to %d = %d*%d*%d colors")
JMESSAGE(JTRC_QUANT_NCOLORS, "Quantizing to %d colors")
JMESSAGE(JTRC_QUANT_SELECTED, "Selected %d colors for quantization")
JMESSAGE(JTRC_RECOVERY_ACTION, "At marker 0x%02x, recovery action %d")
JMESSAGE(JTRC_RST, "RST%d")
JMESSAGE(JTRC_SMOOTH_NOTIMPL,
    "Smoothing not supported with nonstandard sampling ratios")
JMESSAGE(JTRC_SOF, "Start Of Frame 0x%02x: width=%u, height=%u, components=%d")
JMESSAGE(JTRC_SOF_COMPONENT, "    Component %d: %dhx%dv q=%d")
JMESSAGE(JTRC_SOI, "Start Of Image")
JMESSAGE(JTRC_SOS, "Start Of Scan: %d components")
JMESSAGE(JTRC_SOS_COMPONENT, "    Component %d: dc=%d ac=%d")
JMESSAGE(JTRC_SOS_PARAMS, "    Ss=%d, Se=%d, Ah=%d, Al=%d")
JMESSAGE(JTRC_TFILE_CLOSE, "Closed temporary file %s")

```

```

JMESSAGE(JTRC_TFILE_OPEN, "Opened temporary file %s")
JMESSAGE(JTRC_THUMB_JPEG,
    "JFIF extension marker: JPEG-compressed thumbnail image, length %u")
JMESSAGE(JTRC_THUMB_PALETTE,
    "JFIF extension marker: palette thumbnail image, length %u")
JMESSAGE(JTRC_THUMB_RGB,
    "JFIF extension marker: RGB thumbnail image, length %u")
JMESSAGE(JTRC_UNKNOWN_IDS,
    "Unrecognized component IDs %d %d %d, assuming YCbCr")
JMESSAGE(JTRC_XMS_CLOSE, "Freed XMS handle %u")
JMESSAGE(JTRC_XMS_OPEN, "Obtained XMS handle %u")
JMESSAGE(JWRN_ADOBE_XFORM, "Unknown Adobe color transform code %d")
JMESSAGE(JWRN_BOGUS_PROGRESSION,
    "Inconsistent progression sequence for component %d coefficient %d")
JMESSAGE(JWRN_EXTRANEIOUS_DATA,
    "Corrupt JPEG data: %u extraneous bytes before marker 0x%02x")
JMESSAGE(JWRN_HIT_MARKER, "Corrupt JPEG data: premature end of data segment")
JMESSAGE(JWRN_HUFF_BAD_CODE, "Corrupt JPEG data: bad Huffman code")
JMESSAGE(JWRN_JFIF_MAJOR, "Warning: unknown JFIF revision number %d.%02d")
JMESSAGE(JWRN_JPEG_EOF, "Premature end of JPEG file")
JMESSAGE(JWRN_MUST_RESYNC,
    "Corrupt JPEG data: found marker 0x%02x instead of RST%d")
JMESSAGE(JWRN_NOT_SEQUENTIAL, "Invalid SOS parameters for sequential JPEG")
JMESSAGE(JWRN_TOO_MUCH_DATA, "Application transferred too many scanlines")

```

```

#ifdef JMAKE_ENUM_LIST

```

```

    JMSG_LASTMSGCODE
} J_MESSAGE_CODE;

```

```

#undef JMAKE_ENUM_LIST
#endif /* JMAKE_ENUM_LIST */

```

```

/* Zap JMESSAGE macro so that future re-inclusions do nothing by default */
#undef JMESSAGE

```

```

#ifndef JERROR_H
#define JERROR_H

```

```

/* Macros to simplify using the error and trace message stuff */
/* The first parameter is either type of cinfo pointer */

```

```

/* Fatal errors (print message and exit) */

```

```

#define ERREXIT(cinfo,code) \
    ((cinfo)->err->msg_code = (code), \
    (*(cinfo)->err->error_exit) ((j_common_ptr) (cinfo)))

```

```

#define ERREXIT1(cinfo,code,p1) \
    ((cinfo)->err->msg_code = (code), \
    (cinfo)->err->msg_parm.i[0] = (p1), \
    (*(cinfo)->err->error_exit) ((j_common_ptr) (cinfo)))

```

```

#define ERREXIT2(cinfo,code,p1,p2) \
    ((cinfo)->err->msg_code = (code), \
    (cinfo)->err->msg_parm.i[0] = (p1), \
    (cinfo)->err->msg_parm.i[1] = (p2), \
    (*(cinfo)->err->error_exit) ((j_common_ptr) (cinfo)))

```

```

#define ERREXIT3(cinfo,code,p1,p2,p3) \
    ((cinfo)->err->msg_code = (code), \
    (cinfo)->err->msg_parm.i[0] = (p1), \
    (cinfo)->err->msg_parm.i[1] = (p2), \
    (cinfo)->err->msg_parm.i[2] = (p3), \
    (*(cinfo)->err->error_exit) ((j_common_ptr) (cinfo)))

```

```

#define ERREXIT4(cinfo,code,p1,p2,p3,p4) \
    ((cinfo)->err->msg_code = (code), \
    (cinfo)->err->msg_parm.i[0] = (p1), \
    (cinfo)->err->msg_parm.i[1] = (p2), \
    (cinfo)->err->msg_parm.i[2] = (p3), \
    (cinfo)->err->msg_parm.i[3] = (p4), \
    (*(cinfo)->err->error_exit) ((j_common_ptr) (cinfo)))

```

```

#define ERREXITS(cinfo,code,str) \
    ((cinfo)->err->msg_code = (code), \
    strncpy((cinfo)->err->msg_parm.s, (str), JMSG_STR_PARM_MAX), \
    (*(cinfo)->err->error_exit) ((j_common_ptr) (cinfo)))

```

```

#define MAKESTMT(stuff)    do { stuff } while (0)

```

```

/* Nonfatal errors (we can keep going, but the data is probably corrupt) */

```

```

#define WARNMS(cinfo,code) \
    ((cinfo)->err->msg_code = (code), \
    (*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), -1))

```

```

#define WARNMS1(cinfo,code,p1) \
    ((cinfo)->err->msg_code = (code), \
    (cinfo)->err->msg_parm.i[0] = (p1), \
    (*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), -1))
#define WARNMS2(cinfo,code,p1,p2) \
    ((cinfo)->err->msg_code = (code), \
    (cinfo)->err->msg_parm.i[0] = (p1), \
    (cinfo)->err->msg_parm.i[1] = (p2), \
    (*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), -1))

/* Informational/debugging messages */
#define TRACEMS(cinfo,lv1,code) \
    ((cinfo)->err->msg_code = (code), \
    (*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)))
#define TRACEMS1(cinfo,lv1,code,p1) \
    ((cinfo)->err->msg_code = (code), \
    (cinfo)->err->msg_parm.i[0] = (p1), \
    (*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)))
#define TRACEMS2(cinfo,lv1,code,p1,p2) \
    ((cinfo)->err->msg_code = (code), \
    (cinfo)->err->msg_parm.i[0] = (p1), \
    (cinfo)->err->msg_parm.i[1] = (p2), \
    (*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)))
#define TRACEMS3(cinfo,lv1,code,p1,p2,p3) \
    MAKESTMT(int * _mp = (cinfo)->err->msg_parm.i; \
    _mp[0] = (p1); _mp[1] = (p2); _mp[2] = (p3); \
    (cinfo)->err->msg_code = (code); \
    (*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)); )
#define TRACEMS4(cinfo,lv1,code,p1,p2,p3,p4) \
    MAKESTMT(int * _mp = (cinfo)->err->msg_parm.i; \
    _mp[0] = (p1); _mp[1] = (p2); _mp[2] = (p3); _mp[3] = (p4); \
    (cinfo)->err->msg_code = (code); \
    (*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)); )
#define TRACEMS5(cinfo,lv1,code,p1,p2,p3,p4,p5) \
    MAKESTMT(int * _mp = (cinfo)->err->msg_parm.i; \
    _mp[0] = (p1); _mp[1] = (p2); _mp[2] = (p3); _mp[3] = (p4); \
    _mp[4] = (p5); \
    (cinfo)->err->msg_code = (code); \
    (*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)); )
#define TRACEMS8(cinfo,lv1,code,p1,p2,p3,p4,p5,p6,p7,p8) \
    MAKESTMT(int * _mp = (cinfo)->err->msg_parm.i; \
    _mp[0] = (p1); _mp[1] = (p2); _mp[2] = (p3); _mp[3] = (p4); \
    _mp[4] = (p5); _mp[5] = (p6); _mp[6] = (p7); _mp[7] = (p8); \
    (cinfo)->err->msg_code = (code); \
    (*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)); )
#define TRACEMSS(cinfo,lv1,code,str) \
    ((cinfo)->err->msg_code = (code), \
    strncpy((cinfo)->err->msg_parm.s, (str), JMSG_STR_PARM_MAX), \
    (*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)))

#endif /* JERROR_H */

```

```

/*
 * jinclude.h
 *
 * Copyright (C) 1991-1994, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file exists to provide a single place to fix any problems with
 * including the wrong system include files. (Common problems are taken
 * care of by the standard jconfig symbols, but on really weird systems
 * you may have to edit this file.)
 *
 * NOTE: this file is NOT intended to be included by applications using the
 * JPEG library. Most applications need only include jpeglib.h.
 */

/* Include auto-config file to find out which system include files we need. */

#include "jconfig.h"      /* auto configuration options */
#define JCONFIG_INCLUDED  /* so that jpeglib.h doesn't do it again */

/*
 * We need the NULL macro and size_t typedef.
 * On an ANSI-conforming system it is sufficient to include <stddef.h>.
 * Otherwise, we get them from <stdlib.h> or <stdio.h>; we may have to
 * pull in <sys/types.h> as well.
 * Note that the core JPEG library does not require <stdio.h>;
 * only the default error handler and data source/destination modules do.
 * But we must pull it in because of the references to FILE in jpeglib.h.
 * You can remove those references if you want to compile without <stdio.h>.
 */
#ifdef HAVE_STDDEF_H
#include <stddef.h>
#endif
#ifdef HAVE_STDLIB_H
#include <stdlib.h>
#endif
#ifdef NEED_SYS_TYPES_H
#include <sys/types.h>
#endif
#include <stdio.h>

/*
 * We need memory copying and zeroing functions, plus strncpy().
 * ANSI and System V implementations declare these in <string.h>.
 * BSD doesn't have the mem() functions, but it does have bcopy()/bzero().
 * Some systems may declare memset and memcpy in <memory.h>.
 *
 * NOTE: we assume the size parameters to these functions are of type size_t.
 * Change the casts in these macros if not!
 */

#ifdef NEED_BSD_STRINGS
#include <strings.h>
#define MEMZERO(target,size)  bzero((void *) (target), (size_t) (size))
#define MEMCOPY(dest,src,size) bcopy((const void *) (src), (void *) (dest), (size_t) (size))
#else /* not BSD, assume ANSI/SysV string lib */
#include <string.h>
#define MEMZERO(target,size)  memset((void *) (target), 0, (size_t) (size))
#define MEMCOPY(dest,src,size) memcpy((void *) (dest), (const void *) (src), (size_t) (size))
#endif

/*
 * In ANSI C, and indeed any rational implementation, size_t is also the
 * type returned by sizeof(). However, it seems there are some irrational
 * implementations out there, in which sizeof() returns an int even though
 * size_t is defined as long or unsigned long. To ensure consistent results
 * we always use this SIZEOF() macro in place of using sizeof() directly.
 */
#define SIZEOF(object) ((size_t) sizeof(object))

```



```

/*
 * The modules that use fread() and fwrite() always invoke them through
 * these macros. On some systems you may need to twiddle the argument casts.
 * CAUTION: argument order is different from underlying functions!
 */

#define JFREAD(file,buf,sizeofbuf) \
    ((size_t) fread((void *) (buf), (size_t) 1, (size_t) (sizeofbuf), (file)))
#define JFWRITE(file,buf,sizeofbuf) \
    ((size_t) fwrite((const void *) (buf), (size_t) 1, (size_t) (sizeofbuf), (file)))

```

```

/*
 * jmemsys.h
 *
 * Copyright (C) 1992-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This include file defines the interface between the system-independent
 * and system-dependent portions of the JPEG memory manager.  No other
 * modules need include it.  (The system-independent portion is jmemmgr.c;
 * there are several different versions of the system-dependent portion.)
 *
 * This file works as-is for the system-dependent memory managers supplied
 * in the IJG distribution.  You may need to modify it if you write a
 * custom memory manager.  If system-dependent changes are needed in
 * this file, the best method is to #ifdef them based on a configuration
 * symbol supplied in jconfig.h, as we have done with USE_MSDOS_MEMMGR
 * and USE_MAC_MEMMGR.
 */

/* Short forms of external names for systems with brain-damaged linkers. */

#ifdef NEED_SHORT_EXTERNAL_NAMES
#define jpeg_get_small      jGetSmall
#define jpeg_free_small    jFreeSmall
#define jpeg_get_large     jGetLarge
#define jpeg_free_large    jFreeLarge
#define jpeg_mem_available jMemAvail
#define jpeg_open_backing_store jOpenBackStore
#define jpeg_mem_init      jMemInit
#define jpeg_mem_term      jMemTerm
#endif /* NEED_SHORT_EXTERNAL_NAMES */

/*
 * These two functions are used to allocate and release small chunks of
 * memory.  (Typically the total amount requested through jpeg_get_small is
 * no more than 20K or so; this will be requested in chunks of a few K each.)
 * Behavior should be the same as for the standard library functions malloc
 * and free; in particular, jpeg_get_small must return NULL on failure.
 * On most systems, these ARE malloc and free.  jpeg_free_small is passed the
 * size of the object being freed, just in case it's needed.
 * On an 80x86 machine using small-data memory model, these manage near heap.
 */
EXTERN(void *) jpeg_get_small JPP((j_common_ptr cinfo, size_t sizeofobject));
EXTERN(void) jpeg_free_small JPP((j_common_ptr cinfo, void * object,
                                  size_t sizeofobject));

/*
 * These two functions are used to allocate and release large chunks of
 * memory (up to the total free space designated by jpeg_mem_available).
 * The interface is the same as above, except that on an 80x86 machine,
 * far pointers are used.  On most other machines these are identical to
 * the jpeg_get/free_small routines; but we keep them separate anyway,
 * in case a different allocation strategy is desirable for large chunks.
 */
EXTERN(void *) jpeg_get_large JPP((j_common_ptr cinfo,
                                   size_t sizeofobject));
EXTERN(void) jpeg_free_large JPP((j_common_ptr cinfo, void * object,
                                  size_t sizeofobject));

/*
 * The macro MAX_ALLOC_CHUNK designates the maximum number of bytes that may
 * be requested in a single call to jpeg_get_large (and jpeg_get_small for that
 * matter, but that case should never come into play).  This macro is needed
 * to model the 64Kb-segment-size limit of far addressing on 80x86 machines.
 * On those machines, we expect that jconfig.h will provide a proper value.
 * On machines with 32-bit flat address spaces, any large constant may be used.
 *
 * NB: jmemmgr.c expects that MAX_ALLOC_CHUNK will be representable as type
 * size_t and will be a multiple of sizeof(aligned_type).
 */

#ifndef MAX_ALLOC_CHUNK /* may be overridden in jconfig.h */
#define MAX_ALLOC_CHUNK 1000000000L
#endif

```

```

/*
 * This routine computes the total space still available for allocation by
 * jpeg_get_large. If more space than this is needed, backing store will be
 * used. NOTE: any memory already allocated must not be counted.
 *
 * There is a minimum space requirement, corresponding to the minimum
 * feasible buffer sizes; jmemmgr.c will request that much space even if
 * jpeg_mem_available returns zero. The maximum space needed, enough to hold
 * all working storage in memory, is also passed in case it is useful.
 * Finally, the total space already allocated is passed. If no better
 * method is available, cinfo->mem->max_memory_to_use - already_allocated
 * is often a suitable calculation.
 *
 * It is OK for jpeg_mem_available to underestimate the space available
 * (that'll just lead to more backing-store access than is really necessary).
 * However, an overestimate will lead to failure. Hence it's wise to subtract
 * a slop factor from the true available space. 5% should be enough.
 *
 * On machines with lots of virtual memory, any large constant may be returned.
 * Conversely, zero may be returned to always use the minimum amount of memory.
 */

```

```

EXTERN(long) jpeg_mem_available JPP((j_common_ptr cinfo,
                                     long min_bytes_needed,
                                     long max_bytes_needed,
                                     long already_allocated));

```

```

/*
 * This structure holds whatever state is needed to access a single
 * backing-store object. The read/write/close method pointers are called
 * by jmemmgr.c to manipulate the backing-store object; all other fields
 * are private to the system-dependent backing store routines.
 */

```

```

#define TEMP_NAME_LENGTH 64 /* max length of a temporary file's name */

```

```

#ifdef USE_MSDOS_MEMMGR /* DOS-specific junk */

```

```

typedef unsigned short XMSH; /* type of extended-memory handles */
typedef unsigned short EMSH; /* type of expanded-memory handles */

```

```

typedef union {
    short file_handle; /* DOS file handle if it's a temp file */
    XMSH xms_handle; /* handle if it's a chunk of XMS */
    EMSH ems_handle; /* handle if it's a chunk of EMS */
} handle_union;

```

```

#endif /* USE_MSDOS_MEMMGR */

```

```

#ifdef USE_MAC_MEMMGR /* Mac-specific junk */
#include <Files.h>
#endif /* USE_MAC_MEMMGR */

```

```

typedef struct backing_store_struct * backing_store_ptr;

```

```

typedef struct backing_store_struct {
    /* Methods for reading/writing/closing this backing-store object */
    JMETHOD(void, read_backing_store, (j_common_ptr cinfo,
                                       backing_store_ptr info,
                                       void * buffer_address,
                                       long file_offset, long byte_count));
    JMETHOD(void, write_backing_store, (j_common_ptr cinfo,
                                       backing_store_ptr info,
                                       void * buffer_address,
                                       long file_offset, long byte_count));
    JMETHOD(void, close_backing_store, (j_common_ptr cinfo,
                                       backing_store_ptr info));

```

```

    /* Private fields for system-dependent backing-store management */
#ifdef USE_MSDOS_MEMMGR
    /* For the MS-DOS manager (jmemdos.c), we need: */
    handle_union handle; /* reference to backing-store storage object */
    char temp_name[TEMP_NAME_LENGTH]; /* name if it's a file */
#else
#ifdef USE_MAC_MEMMGR
    /* For the Mac manager (jmemmac.c), we need: */
    short temp_file; /* file reference number to temp file */

```

```

FSSpec tempSpec; /* the FSSpec for the temp file */
char temp_name[TEMP_NAME_LENGTH]; /* name if it's a file */
#else
/* For a typical implementation with temp files, we need: */
FILE * temp_file; /* stdio reference to temp file */
char temp_name[TEMP_NAME_LENGTH]; /* name of temp file */
#endif
#endif
} backing_store_info;

```

```

/*
 * Initial opening of a backing-store object. This must fill in the
 * read/write/close pointers in the object. The read/write routines
 * may take an error exit if the specified maximum file size is exceeded.
 * (If jpeg_mem_available always returns a large value, this routine can
 * just take an error exit.)
 */

```

```

EXTERN(void) jpeg_open_backing_store JPP((j_common_ptr cinfo,
      backing_store_ptr info,
      long total_bytes_needed));

```

```

/*
 * These routines take care of any system-dependent initialization and
 * cleanup required. jpeg_mem_init will be called before anything is
 * allocated (and, therefore, nothing in cinfo is of use except the error
 * manager pointer). It should return a suitable default value for
 * max_memory_to_use; this may subsequently be overridden by the surrounding
 * application. (Note that max_memory_to_use is only important if
 * jpeg_mem_available chooses to consult it ... no one else will.)
 * jpeg_mem_term may assume that all requested memory has been freed and that
 * all opened backing-store objects have been closed.
 */

```

```

EXTERN(long) jpeg_mem_init JPP((j_common_ptr cinfo));
EXTERN(void) jpeg_mem_term JPP((j_common_ptr cinfo));

```



```

    * On nearly all machines "short" will do nicely.
    */

typedef short JSAMPLE;
#define GETJSAMPLE(value) ((int) (value))

#define MAXJSAMPLE 4095
#define CENTERJSAMPLE 2048

#endif /* BITS_IN_JSAMPLE == 12 */

/* Representation of a DCT frequency coefficient.
 * This should be a signed value of at least 16 bits; "short" is usually OK.
 * Again, we allocate large arrays of these, but you can change to int
 * if you have memory to burn and "short" is really slow.
 */

typedef short JCOEF;

/* Compressed datastreams are represented as arrays of JOCTET.
 * These must be EXACTLY 8 bits wide, at least once they are written to
 * external storage. Note that when using the stdio data source/destination
 * managers, this is also the data type passed to fread/fwrite.
 */

#ifdef HAVE_UNSIGNED_CHAR

typedef unsigned char JOCTET;
#define GETJOCTET(value) (value)

#else /* not HAVE_UNSIGNED_CHAR */

typedef char JOCTET;
#ifdef CHAR_IS_UNSIGNED
#define GETJOCTET(value) (value)
#else
#define GETJOCTET(value) ((value) & 0xFF)
#endif /* CHAR_IS_UNSIGNED */

#endif /* HAVE_UNSIGNED_CHAR */

/*
 * These typedefs are used for various table entries and so forth.
 * They must be at least as wide as specified; but making them too big
 * won't cost a huge amount of memory, so we don't provide special
 * extraction code like we did for JSAMPLE. (In other words, these
 * typedefs live at a different point on the speed/space tradeoff curve.)
 */

/*
 * UINT8 must hold at least the values 0..255. */

#ifdef HAVE_UNSIGNED_CHAR
typedef unsigned char UINT8;
#else /* not HAVE_UNSIGNED_CHAR */
#ifdef CHAR_IS_UNSIGNED
typedef char UINT8;
#else /* not CHAR_IS_UNSIGNED */
typedef short UINT8;
#endif /* CHAR_IS_UNSIGNED */
#endif /* HAVE_UNSIGNED_CHAR */

/*
 * UINT16 must hold at least the values 0..65535. */

#ifdef HAVE_UNSIGNED_SHORT
typedef unsigned short UINT16;
#else /* not HAVE_UNSIGNED_SHORT */
typedef unsigned int UINT16;
#endif /* HAVE_UNSIGNED_SHORT */

/*
 * INT16 must hold at least the values -32768..32767. */

#ifdef HAVE_SHORT
typedef short INT16;
#else
typedef int INT16;
#endif

/*
 * INT32 must hold at least signed 32-bit values. */

#ifdef HAVE_INT32
typedef int INT32;
#else
typedef long INT32;
#endif

```

```

typedef int INT32;
#endif

/* Datatype used for image dimensions. The JPEG standard only supports
 * images up to 64K*64K due to 16-bit fields in SOF markers. Therefore
 * "unsigned int" is sufficient on all machines. However, if you need to
 * handle larger images and you don't mind deviating from the spec, you
 * can change this datatype.
 */

typedef unsigned int JDIMENSION;

#define JPEG_MAX_DIMENSION 65500L /* a tad under 64K to prevent overflows */

/* These macros are used in all function definitions and extern declarations.
 * You could modify them if you need to change function linkage conventions;
 * in particular, you'll need to do that to make the library a Windows DLL.
 * Another application is to make all functions global for use with debuggers
 * or code profilers that require it.
 */

/* a function called through method pointers: */
#define METHODDEF(type) static type
/* a function used only in its module: */
#define LOCAL(type) static type
/* a function referenced thru EXTERNS: */
#define GLOBAL(type) type
/* a reference to a GLOBAL function: */
#define EXTERN(type) extern type

/* This macro is used to declare a "method", that is, a function pointer.
 * We want to supply prototype parameters if the compiler can cope.
 * Note that the arglist parameter must be parenthesized!
 * Again, you can customize this if you need special linkage keywords.
 */
#ifdef HAVE_PROTOTYPES
#define JMETHOD(type,methodname,arglist) type (*methodname) arglist
#else
#define JMETHOD(type,methodname,arglist) type (*methodname) ()
#endif

/* Here is the pseudo-keyword for declaring pointers that must be "far"
 * on 80x86 machines. Most of the specialized coding for 80x86 is handled
 * by just saying "FAR *" where such a pointer is needed. In a few places
 * explicit coding is needed; see uses of the NEED_FAR_POINTERS symbol.
 */
/*### Intiaz : commented this out.
#ifdef NEED_FAR_POINTERS
#define FAR far
#else
#define FAR far
#endif
*/

/*
 * On a few systems, type boolean and/or its values FALSE, TRUE may appear
 * in standard header files. Or you may have conflicts with application-
 * specific header files that you want to include together with these files.
 * Defining HAVE_BOOLEAN before including jpeglib.h should make it work.
 */
#ifdef HAVE_BOOLEAN
typedef int boolean;
#endif
#ifndef FALSE /* in case these macros already exist */
#define FALSE 0 /* values of boolean */
#endif
#ifndef TRUE
#define TRUE 1
#endif

/*
 * The remaining options affect code selection within the JPEG library,

```

```

* but they don't need to be visible to most applications using the library.
* To minimize application namespace pollution, the symbols won't be
* defined unless JPEG_INTERNALS or JPEG_INTERNAL_OPTIONS has been defined.
*/

```

```

#ifdef JPEG_INTERNALS
#define JPEG_INTERNAL_OPTIONS
#endif

```

```

#ifdef JPEG_INTERNAL_OPTIONS

```

```

/*
* These defines indicate whether to include various optional functions.
* Undefined some of these symbols will produce a smaller but less capable
* library. Note that you can leave certain source files out of the
* compilation/linking process if you've #undef'd the corresponding symbols.
* (You may HAVE to do that if your compiler doesn't like null source files.)
*/

```

```

/* Arithmetic coding is unsupported for legal reasons. Complaints to IBM. */

```

```

/* Capability options common to encoder and decoder: */

```

```

#define DCT_ISLOW_SUPPORTED /* slow but accurate integer algorithm */
#define DCT_IFAST_SUPPORTED /* faster, less accurate integer method */
#define DCT_FLOAT_SUPPORTED /* floating-point: accurate, fast on fast HW */

```

```

/* Encoder capability options: */

```

```

#undef C_ARITH_CODING_SUPPORTED /* Arithmetic coding back end? */
#define C_MULTISCAN_FILES_SUPPORTED /* Multiple-scan JPEG files? */
#define C_PROGRESSIVE_SUPPORTED /* Progressive JPEG? (Requires MULTISCAN)*/
#define ENTROPY_OPT_SUPPORTED /* Optimization of entropy coding parms? */
/* Note: if you selected 12-bit data precision, it is dangerous to turn off
ENTROPY_OPT_SUPPORTED. The standard Huffman tables are only good for 8-bit
precision, so jchuff.c normally uses entropy optimization to compute
usable tables for higher precision. If you don't want to do optimization,
you'll have to supply different default Huffman tables.
The exact same statements apply for progressive JPEG: the default tables
don't work for progressive mode. (This may get fixed, however.)
*/
#define INPUT_SMOOTHING_SUPPORTED /* Input image smoothing option? */

```

```

/* Decoder capability options: */

```

```

#undef D_ARITH_CODING_SUPPORTED /* Arithmetic coding back end? */
#define D_MULTISCAN_FILES_SUPPORTED /* Multiple-scan JPEG files? */
#define D_PROGRESSIVE_SUPPORTED /* Progressive JPEG? (Requires MULTISCAN)*/
#define SAVE_MARKERS_SUPPORTED /* jpeg_save_markers() needed? */
#define BLOCK_SMOOTHING_SUPPORTED /* Block smoothing? (Progressive only) */
#define IDCT_SCALING_SUPPORTED /* Output rescaling via IDCT? */
#undef UPSAMPLE_SCALING_SUPPORTED /* Output rescaling at upsample stage? */
#define UPSAMPLE_MERGING_SUPPORTED /* Fast path for sloppy upsampling? */
#define QUANT_1PASS_SUPPORTED /* 1-pass color quantization? */
#define QUANT_2PASS_SUPPORTED /* 2-pass color quantization? */

```

```

/* more capability options later, no doubt */

```

```

/*
* Ordering of RGB data in scanlines passed to or from the application.
* If your application wants to deal with data in the order B,G,R, just
* change these macros. You can also deal with formats such as R,G,B,X
* (one extra byte per pixel) by changing RGB_PIXELSIZE. Note that changing
* the offsets will also change the order in which colormap data is organized.
* RESTRICTIONS:
* 1. The sample applications cjpeg,djpeg do NOT support modified RGB formats.
* 2. These macros only affect RGB<=>YCbCr color conversion, so they are not
* useful if you are using JPEG color spaces other than YCbCr or grayscale.
* 3. The color quantizer modules will not behave desirably if RGB_PIXELSIZE
* is not 3 (they don't understand about dummy color components!). So you
* can't use color quantization if you change that value.
*/

```

```

#define RGB_RED 0 /* Offset of Red in an RGB scanline element */
#define RGB_GREEN 1 /* Offset of Green */
#define RGB_BLUE 2 /* Offset of Blue */
#define RGB_PIXELSIZE 3 /* JSAMPLEs per RGB scanline element */

```



```

/* Definitions for speed-related optimizations. */

/* If your compiler supports inline functions, define INLINE
 * as the inline keyword; otherwise define it as empty.
 */

#ifndef INLINE
#ifdef __GNUC__ /* for instance, GNU C knows about inline */
#define INLINE __inline__
#endif
#endif
#ifndef INLINE
#define INLINE /* default is to define it as empty */
#endif
#endif

/* On some machines (notably 68000 series) "int" is 32 bits, but multiplying
 * two 16-bit shorts is faster than multiplying two ints. Define MULTIPLIER
 * as short on such a machine. MULTIPLIER must be at least 16 bits wide.
 */

#ifndef MULTIPLIER
#define MULTIPLIER int /* type for fastest integer multiply */
#endif

/* FAST_FLOAT should be either float or double, whichever is done faster
 * by your compiler. (Note that this type is only used in the floating point
 * DCT routines, so it only matters if you've defined DCT_FLOAT_SUPPORTED.)
 * Typically, float is faster in ANSI C compilers, while double is faster in
 * pre-ANSI compilers (because they insist on converting to double anyway).
 * The code below therefore chooses float if we have ANSI-style prototypes.
 */

#ifndef FAST_FLOAT
#ifdef HAVE_PROTOTYPES
#define FAST_FLOAT float
#else
#define FAST_FLOAT double
#endif
#endif

#endif /* JPEG_INTERNAL_OPTIONS */

```

```

    for(i=1;i<4;i++) r_Leye[i]=-fabs(r_Leye[i]);
}
v1=sqrt(r_Leye[1]*r_Leye[1]+r_Leye[2]*r_Leye[2]+r_Leye[3]*r_Leye[3]);
for (i=1;i<4;i++) { r_Leye[i]/=v1 ; LL[i]=1/r_Leye[i]; }
if(how=='Y') LLL=LL[1]/(r_ALscale[1]*LL[1]);
v3=hypot(r_Leye[1],r_Leye[2]) ; v2=-r_Leye[2]*r_Leye[3]/v3 ; v1=-r_Leye[1]*r_Leye[3]/v3 ;
u1=r_Leye[2]/v3 ; u2=-r_Leye[1]/v3 ;
a=Na[2]*u2-Na[1]*u1 ; b=Na[3]*v3-Na[2]*v2-Na[1]*v1;
m1=(int)(__min(r_LE/r_DIM,r_HI*a/(b*r_DIM))) ; m2=(int)(m1*b/a) ; du=__min(a/m1,b/m2) ;
a=r_DIM/du ; un1=u1*Na[1]*a ; un2=u2*Na[2]*a ;
vn1=v1*Na[1]*a ; vn2=v2*Na[2]*a ; vn3=v3*Na[3]*a ;

NX[0]=0;                NX[1]=(int)(r_HI+vn2);
NX[2]=0;                NX[3]=(int)(r_HI+vn2-vn3);
NX[4]=(int)(-un1);      NX[5]=(int)(r_HI+vn1+vn2-vn3);
NX[6]=(int)(m1*r_DIM-1); NX[7]=(int)(r_HI-v1*(NX[6]-un2)/u1-vn3);
NX[8]=NX[6];           NX[9]=(int)(r_HI-v1*(NX[8]-un2)/u1);
NX[10]=(int)(un2);      NX[11]=(int)(r_HI);
NX[12]=NX[0];          NX[13]=NX[1];
a=__max((double)(r_HI)/(r_MAXY-25),(double)(r_LE)/(r_MAXX-225)) ;
for(i=1;i<=13;i+=2)
{
    nx[i]=(int)(NX[i]/a+20);
    nx[i-1]=(int)(NX[i-1]/a+220);
}
//!!setfillstyle(EMPTY_FILL,0) ; bar(214,17,r_MAXX,r_MAXY);
if ((m1<=3)|| (m2<=3)|| (du<0.001))
{
    sprintf(r_error,"Rendered image is too small");
    return false;
}
//!!setlinestyle(SOLID_LINE,0,NORM_WIDTH); rectangle (216,19,224+r_LE/a,24+r_HI/a);
//!!drawpoly(7,nx);
b=220-un1/a; c=20+(r_HI+vn1+vn2)/a;
//!!line(b,c,nx[0],nx[1]); line(b,c,nx[4],nx[5]);line(b,c,nx[8],nx[9]);
//!!bar(b,c,b+8,c+8); bar(nx[0],nx[1],nx[0]+16,nx[1]+8);
//!!bar(nx[4],nx[5],nx[4]+16,nx[5]+8); bar(nx[8]-8,nx[9],nx[8]+8,nx[9]+8);
//!!outtextxy(b,c,"0");
//!!if(r_OZ>0) outtextxy(nx[4],nx[5],"z"); else outtextxy(nx[4],nx[5],"-z");
//!!switch(r_OX) {
//!!case 1: outtextxy(nx[0],nx[1],"x"); outtextxy(nx[8]-8,nx[9],"y"); break;
//!!case -1: outtextxy(nx[0],nx[1],"-x"); outtextxy(nx[8]-8,nx[9],"-y"); break;
//!!case 2: outtextxy(nx[0],nx[1],"y"); outtextxy(nx[8]-8,nx[9],"-x"); break;
//!!case -2: outtextxy(nx[0],nx[1],"-y"); outtextxy(nx[8]-8,nx[9],"x"); break;
//!!}
b=220+un2/a; c=20+(r_HI-vn3)/a;
//!!line (b,c,220+un2/a,20+r_HI/a); line (b,c,220,20+(r_HI+vn2-vn3)/a);
//!!line (b,c,nx[8],nx[7]); setlinestyle(SOLID_LINE,0,THICK_WIDTH);
CU1=u1; CU2=u2; CV1=v1; CV2=v2; CV3=v3;
AP[1]=du; AP[2]=un1; AP[3]=un2;
CF1=vn1; CF2=vn2; CF3=vn3;
r_DIM1=m1; r_DIM2=m2;
return true;
}

/*****
*
* Find cube sign with respect to the current surface level.
* Returns: 0-equal signs of the cube vertices
*          1-different signs
*
*****/
bool RayTracer::CUBESIGN(int a1, int a2, int a3, int a4, int a5, int a6, int a7, int a8)
{
    if(
        ((a1>r_LEV)&&(a2>r_LEV)&&(a3>r_LEV)&&(a4>r_LEV)&&
         (a5>r_LEV)&&(a6>r_LEV)&&(a7>r_LEV)&&(a8>r_LEV)) ||
        ((a1<r_LEV)&&(a2<r_LEV)&&(a3<r_LEV)&&(a4<r_LEV)&&
         (a5<r_LEV)&&(a6<r_LEV)&&(a7<r_LEV)&&(a8<r_LEV))) return false;
}

```

```

else return true;
}

```

```

/*****
*
*   MOVING RAY, JUMPING FROM ONE EDGE TO ANOTHER,
*   AND SEEKING FOR ROOT, IF IT EXISTS.
*   RETURNS: DEPTH>=0 - BODY'S FOUND.
*             -1 - BODY WASN'T FOUND
*
*****/
double RayTracer::JUMP(double t)
{
    int    m,k,k1,i,il,fin,h,sp[4];
    long    pfh,qfh,pd,qd;
    long    dt=0;
    double  d,f,fh,a,b;

    k1=0;
    /* LET'S JUMP : */
    jump: i=TOP[1] ; k=TOP[2] ; m=TOP[3] ;
    if (CUBESIGN(F(i,k,m),F(i,k+1,m),F(i,k,m+1),F(i,k+1,m+1),F(i+1,k,m),
        F(i+1,k+1,m),F(i+1,k,m+1),F(i+1,k+1,m+1))=0)
    {
        /* JUMP'S STEP: */
        h=__min(__min(P[1],P[2]),P[3]) ;
        for(il=1;il<4;il++) {
            if(h==P[il]) { if(P[il]==-IL[il]) NX[il]--;
                else P[il]=-IL[il] ;
                if(NX[il]<=0) return (-1);
                TOP[il]=NX[il]-1; IED[il]=il;
            }
        }
        else { if(P[il]==-IL[il]) { if(NX[il]<1) return (-1) ;
            NX[il]--;
            }
            P[il]-=h ; IED[il]=0; TOP[il]=NX[il] ;
        } /* next il */
        k1=0 ;
        /* end of going trough cube with equal signs of tops */
    }
    else
    {
        if(k1!=0) { pd=pfh; qd=qfh; }
        else { EDGE() ; pd=PE; qd=QE; if(pd==0) return ( t+dt); }
        fin=0 ;

        /* JUMP'S STEP: */
        h=__min(__min(P[1],P[2]),P[3]) ;
        for(il=1;il<4;il++) {
            sp[il]=P[il];
            if(h==P[il]) { if(P[il]==-IL[il]) NX[il]--;
                else P[il]=-IL[il] ;
                if(NX[il]<=0) fin=1;
                TOP[il]=NX[il]-1; IED[il]=il;
            }
        }
        else { if(P[il]==-IL[il]) { if(NX[il]<1) return (-1);
            NX[il]--;
            }
            P[il]-=h ; IED[il]=0 ; TOP[il]=NX[il] ; sp[il]+=P[il];
        } /* next il */
        EDGE(); pfh=PE; qfh=QE; /* pfh/qfh = PE/QE */
        if( ((pfh>0)&&(pd<0)) || ((pfh<0)&&(pd>0)) ) {
            f=FUNC(sp[1],sp[2],sp[3]); d=(double)(pd)/qd ; fh=(double)(pfh)/qfh ;
            if(f==fh) return ( t+dt+0.5*h*d/(d-f) );
            if(f==d) return ( t+dt+h*(f-0.5*fh)/(f-fh) );
            a=f/(fh-d); b=d-f; b+=b; b=fh/b;
            return ( t+dt+(a+b)*h*d/(fh-f) );
        }
    }
}

```

```

    }

if( pfh==0 ) { f=FUNC(sp[1],sp[2],sp[3]);
    if( ((f>=0)&&(pd<0)) || ((f<=0)&&(pd>0)) ) {
        d=(double)(pd)/qd; return ( t+dt+(0.5*h*d)/(d-2*f) );
    }
    else return ( t+(dt+h) );
}

k1=1 ; if(fin==1) return (-1) ;
} /* end of going trough the cube with different signs of tops */

dt+=h;
goto jump;
}

/*****
*
* MOVING RAY, JUMPING FROM ONE EDGE TO ANOTHER,
* AND SEEKING FOR ROOT, IF IT EXISTS.
* RETURNS:      +1 - BODY'S FOUND.
*               -1 - BODY WASN'T FOUND
*
*****/
int RayTracer::Q_JUMP()
{
    int m,k,k1,i,i1,fin,h;
    long fh,d;

    k1=0;
    /* LET'S Q_JUMP : */
    qjump: i=TOP[1] ; k=TOP[2] ; m=TOP[3] ;
    if( CUBESIGN(F(i,k,m),F(i,k+1,m),F(i,k,m+1),F(i,k+1,m+1),F(i+1,k,m),
        F(i+1,k+1,m),F(i+1,k,m+1),F(i+1,k+1,m+1))!=0)
    {
        /* Q_JUMP'S STEP: */
        h=_min(_min(P[1],P[2]),P[3]) ;
        for(i1=1;i1<4;i1++) {
            if(h==P[i1]) { if(P[i1]==-IL[i1]) NX[i1]--;
                else P[i1]=-IL[i1] ;
                if(NX[i1]<=0) return (-1);
                TOP[i1]=NX[i1]-1; IED[i1]=i1;
            }
            else { if(P[i1]==-IL[i1]) { if(NX[i1]<1) return (-1) ;
                NX[i1]--;
            }
            P[i1]-=h ; IED[i1]=0; TOP[i1]=NX[i1] ;
        }
        /* next i1 */

        k1=0 ;
    } /* end of going trough cube with equal signs of tops */

else
{
    if(k1!=0) d=fh; else { EDGE() ; d=PE; }
    fin=0 ;

    /* Q_JUMP'S STEP: */
    h=_min(_min(P[1],P[2]),P[3]) ;
    for(i1=1;i1<4;i1++) {
        if(h==P[i1]) { if(P[i1]==-IL[i1]) NX[i1]--;
            else P[i1]=-IL[i1] ;
            if(NX[i1]<=0) fin=1;
            TOP[i1]=NX[i1]-1; IED[i1]=i1;
        }
        else { if(P[i1]==-IL[i1]) { if(NX[i1]<1) return (-1);

```

```

        NX[i1]--;
    }
    P[i1] -= h; IED[i1] = 0; TOP[i1] = NX[i1];
}
} /* next i1 */
EDGE(); fh = PE;
if(((d >= 0) && (fh <= 0)) || ((d <= 0) && (fh >= 0))) return(+1);
k1 = 1; if(fin == 1) return(-1);
} /* end of going through the cube with different signs of tops */

goto qjump;
}

/*..... */
/* CALCULATES FUNCTION'S VALUE ON THE EDGE : F = PE/QE
   IL12=IL[1]*IL[2] ; IL13=IL[1]*IL[3] ; IL23=IL[2]*IL[3] ; */
void RayTracer::EDGE()
{
    int i,j,k;
    long z,a,b;
    i = NX[1]; j = NX[2]; k = NX[3]; z = F(i,j,k);
    switch (IED[1]+IED[2]-IED[3]) {
    case -1: PE = P[1]*(F(i+1,j,k)-z)-(z-r_LEV)*IL[1]; QE = -IL[1]; break; /* 0 2 3 */
    case -2: PE = P[2]*(F(i,j+1,k)-z)-(z-r_LEV)*IL[2]; QE = -IL[2]; break; /* 1 0 3 */
    case 3: PE = P[3]*(F(i,j,k+1)-z)-(z-r_LEV)*IL[3]; QE = -IL[3]; break; /* 1 2 0 */
    case -3:
        a = (z-F(i+1,j,k)-F(i,j+1,k)+F(i+1,j+1,k))*P[2]-(F(i+1,j,k)-z)*IL[2];
        b = (F(i,j+1,k)-z)*P[2]-(z-r_LEV)*IL[2];
        PE = a*P[1]-b*IL[1]; QE = IL12; break; /* 0 0 3 */
    case 2:
        a = (z-F(i+1,j,k)-F(i,j,k+1)+F(i+1,j,k+1))*P[3]-(F(i+1,j,k)-z)*IL[3];
        b = (F(i,j,k+1)-z)*P[3]-(z-r_LEV)*IL[3];
        PE = a*P[1]-b*IL[1]; QE = IL13; break; /* 0 2 0 */
    case 1:
        a = (z-F(i,j+1,k)-F(i,j,k+1)+F(i,j+1,k+1))*P[3]-(F(i,j+1,k)-z)*IL[3];
        b = (F(i,j,k+1)-z)*P[3]-(z-r_LEV)*IL[3];
        PE = a*P[2]-b*IL[2]; QE = IL23; break; /* 1 0 0 */
    default: PE = z-r_LEV; QE = 1; break;
    }
    return;
}

/*..... */
/* CALCULATES FUNCTION VALUE IN THE POINT (x,y,z)
   RETURNS- FUNCTION VALUE */
double RayTracer::FUNC(long x, long y, long z)
{
    int i,j,k,f,f1;
    long a,b;
    i = NX[1]; j = NX[2]; k = NX[3]; f = F(i,j,k); f1 = F(i,j,k+1);
    a = x*((f+F(i+1,j+1,k)-F(i+1,j,k)-F(i,j+1,k))*y+(F(i+1,j,k)-f)*DL2)
        +DL1*(F(i,j+1,k)-f)*y+(f-r_LEV)*DL2;
    b = x*((f1+F(i+1,j+1,k+1)-F(i+1,j,k+1)-F(i,j+1,k+1))*y+DL2*(F(i+1,j,k+1)-f1))
        +DL1*(F(i,j+1,k+1)-f1)*y+(f1-r_LEV)*DL2;
    return (a*DL1+(b-a)*(z*D2));
}

/*..... */
/* INSTALL YOUR POINT ON THE CUBE EDGE (X,Y,Z- DISTANCES)
   RETURNS: IS[p][q] (IF POINT CAN'T BE INSTALLED,
        FILLS IS[p][q]=-1)
        */
void RayTracer::INST(int p, int q, int u, int v)
{
    int i;
    double a,h,e[4];

```

```

IS[p][q]=-1;
u+=p; v+=q;
e[1]=CU1*u+CV1*v+CF1; e[2]=CU2*u+CV2*v+CF2; e[3]=CV3*v+CF3;
h=__max(__max(e[1],e[2]),e[3]);
for(i=1;i<4;i++) {
    if(h==e[i]) { NX[i]=r_N[i]; P[i]=-IL[i]; IED[i]=i; TOP[i]=r_N[i]-1; }
    else { a=AP[i]+e[i]-h; if(a<0.5) return;
        AA=(long)a; NX[i]=AA/(-IL[i]); P[i]=AA+(long)(NX[i])*IL[i];
        if(P[i]==0) { if(NX[i]==0) { P[i]=1; IED[i]=0; TOP[i]=0; }
            else { P[i]=-IL[i]; IED[i]=i; TOP[i]=NX[i]-1; }
        }
    }
    else { IED[i]=0; TOP[i]=NX[i]; }
}
/* next i */
IS[p][q]=Q_JUMP (); return ;
}

/*..... */
/*CALCULATE INTENSITY:
    b

a <-r_DIM-> t          c

d

RETURN:
VAL=>r_COLb*BD , IF BODY EXISTS
0             IF BODY WAS NOT FOUND
*/
int RayTracer::INTENS(double a, double b, double c, double d, double t)
{
    double nor1,nor2;
    if (t<0) return 0;
    if(a>=0)
    {
        if(c>=0) nor1=c-a;
        else     nor1=2*(t-a);
    }
    else
    {
        if(c>=0) nor1=2*(c-t);
        else     nor1=0;
    }

    if(d>=0)
    {
        if(b>=0) nor2=b-d;
        else     nor2=2*(t-d);
    }
    else
    {
        if(b>=0) nor2=2*(b-t);
        else     nor2=0;
    }
    return ( (int)( B1+B2/sqrt(nor1*nor1+nor2*nor2+DOWN) ) );
}

/*..... */
/*ESTABLISH SURFACE PRESENCE FILLING IS[][] ARRAY
*
*
*
```

```

*/
void RayTracer::CHESS(int uu, int vv, int i1, int i2, int i3, int i4,
                      int j1, int j2, int j3, int j4, int j5, int j6,
                      int j7, int j8)
{
    signed char p,q,p0;
    // Set IS to unidentified "-2"
    p0=IS[0][0];
    for(p=0;p<r_DIM;p++){ for(q=0;q<r_DIM;q++) IS[p][q]=-2; }
    IS[0][0]=p0;
    // Get corners
    INST(0,r_DIM,uu,vv); INST(r_DIM,0,uu,vv); INST(r_DIM,r_DIM,uu,vv);
    // Is DIM small ?
    if(r_DIM<=2)
    {
        GURO2 ('*',i1,i2,i3,i4,j1,j2,j3,j4,j5,j6,j7,j8,uu,vv) ;
        return ;
    }
    // DIM is >2; do the "chess" work
    p0=IS[0][0];
    if( p0==IS[0][r_DIM] && p0==IS[r_DIM][0] && p0==IS[r_DIM][r_DIM] )
    {
        for(p=0;p<r_DIM;p++){ for(q=0;q<r_DIM;q++) IS[p][q]=p0; }
        GURO2 ('*',i1,i2,i3,i4,j1,j2,j3,j4,j5,j6,j7,j8,uu,vv) ;
        return ;
    }
    q=0; p0=0;
    while (q<r_DIM)
    {
        for(p=p0;p<r_DIM;p+=2)
        {
            if(IS[p][q]!=-2) continue;
            else INST(p,q,uu,vv);
        }
        p0=1-p0; q++;
    }
    for(p=1;p<r_DIM;p+=2)
    {
        if(IS[p-1][0]==IS[p][1] && IS[p-1][0]==IS[p+1][0]) IS[p][0]=IS[p-1][0];
        else INST(p,0,uu,vv);
    }
    for(q=1;q<r_DIM;q+=2)
    {
        if(IS[0][q-1]==IS[1][q] && IS[0][q-1]==IS[0][q+1]) IS[0][q]=IS[0][q-1];
        else INST(0,q,uu,vv);
    }
    for(p=(r_DIM%2)+1;p<r_DIM;p+=2)
    {
        if(IS[p-1][r_DIM]==IS[p][r_DIM-1] && IS[p-1][r_DIM]==IS[p+1][r_DIM])
        { IS[p][r_DIM]=IS[p-1][r_DIM]; }
        else INST(p,r_DIM,uu,vv);
    }
    for(q=(r_DIM%2)+1;q<r_DIM;q+=2)
    {
        if(IS[r_DIM][q-1]==IS[r_DIM-1][q] && IS[r_DIM][q-1]==IS[r_DIM][q+1])
        { IS[r_DIM][q]=IS[r_DIM-1][q]; }
        else INST(r_DIM,q,uu,vv);
    }
    q=1; p0=1;
    while (q<r_DIM)
    {
        for(p=p0+1;p<r_DIM;p+=2)
        {
            if(IS[p][q]!=-2) continue;
            if( IS[p-1][q]==IS[p+1][q] && IS[p-1][q]==IS[p][q-1] &&
                IS[p-1][q]==IS[p][q+1]) IS[p][q]=IS[p-1][q];
            else INST(p,q,uu,vv);
        }
        p0=1-p0; q++;
    }
    GURO2 ('*',i1,i2,i3,i4,j1,j2,j3,j4,j5,j6,j7,j8,uu,vv) ; return ;
}

```

```

/*..... */
/* FILLS SQUARE WITH GURO2 INTENSITY:          J1<-r_DIM->J2
MASK:      '-' -LIGHT
           '*' -AS IN IS' CODE                J8      I3      I4      J3

           J7      I1      I2      J4
           (ix,iy)

           J6      J5

RETURNS PICTURE                                */

void RayTracer::GURO2(char mask, long i1, long i2, long i3, long i4,
                      long j1, long j2, long j3, long j4, long j5,
                      long j6, long j7, long j8, int ix, int iy)
{
    int x,y,xx,yy,choix=0;
    long a12,a21,a20,a02,a11,a01,a10,a00,pas,pasy,in,debut,kpas,lpas;
    long kdebut,ldebut,lpasy;
    if(mask=='*') {
        if(i1>0) choix=1 ; if(i2>0) choix+=10 ;
        if(i3>0) choix+=100; if(i4>0) choix+=1000;
        switch(choix) {
            case 1:  i2=i1 ; i3=i1 ; i4=i1 ;
                    if(j1<=0) j1=i1; if(j2<=0) j2=i1; if(j3<=0) j3=i1;
                    if(j4<=0) j4=i1; if(j5<=0) j5=i1; if(j6<=0) j6=i1;
                    if(j7<=0) j7=i1; if(j8<=0) j8=i1;
                    break;
            case 10: i1=i2 ; i3=i2 ; i4=i2 ;
                    if(j1<=0) j1=i2; if(j2<=0) j2=i2; if(j3<=0) j3=i2;
                    if(j4<=0) j4=i2; if(j5<=0) j5=i2; if(j6<=0) j6=i2;
                    if(j7<=0) j7=i2; if(j8<=0) j8=i2;
                    break;
            case 11: i3=i1 ; i4=i2 ;
                    if(j1<=0) j1=i1; if(j2<=0) j2=i2; if(j3<=0) j3=i2;
                    if(j4<=0) j4=__min(i1,i2); if(j5<=0) j5=i2; if(j6<=0) j6=i1;
                    if(j7<=0) j7=__min(i1,i2); if(j8<=0) j8=i1;
                    break;
            case 100: i1=i3 ; i2=i3 ; i4=i3 ;
                    if(j1<=0) j1=i3; if(j2<=0) j2=i3; if(j3<=0) j3=i3;
                    if(j4<=0) j4=i3; if(j5<=0) j5=i3; if(j6<=0) j6=i3;
                    if(j7<=0) j7=i3; if(j8<=0) j8=i3;
                    break;
            case 101: i2=i1 ; i4=i3 ;
                    if(j1<=0) j1=__min(i1,i3); if(j2<=0) j2=i3; if(j3<=0) j3=i3;
                    if(j4<=0) j4=i1; if(j5<=0) j5=i1; if(j6<=0) j6=__min(i1,i3);
                    if(j7<=0) j7=i1; if(j8<=0) j8=i3;
                    break;
            case 110: i1=(i2+i3)/2 ; i4=i1 ;
                    if(j1<=0) j1=i3; if(j2<=0) j2=i2; if(j3<=0) j3=i3;
                    if(j4<=0) j4=i2; if(j5<=0) j5=i2; if(j6<=0) j6=i3;
                    if(j7<=0) j7=i2; if(j8<=0) j8=i3;
                    break;
            case 111: i4=__min(i2,i3) ;
                    if(j1<=0) j1=__min(i1,i3); if(j2<=0) j2=i2; if(j3<=0) j3=i3;
                    if(j4<=0) j4=__min(i1,i2); if(j5<=0) j5=i2; if(j6<=0) j6=__min(i1,i3);
                    if(j7<=0) j7=__min(i1,i2); if(j8<=0) j8=i3;
                    break;
            case 1000: i1=i4 ; i2=i4 ; i3=i4 ;
                    if(j1<=0) j1=i4; if(j2<=0) j2=i4; if(j3<=0) j3=i4;
                    if(j4<=0) j4=i4; if(j5<=0) j5=i4; if(j6<=0) j6=i4;
                    if(j7<=0) j7=i4; if(j8<=0) j8=i4;
                    break;
            case 1001: i2=(i1+i4)/2 ; i3=i2 ;
                    if(j1<=0) j1=i1; if(j2<=0) j2=i4; if(j3<=0) j3=i4;
                    if(j4<=0) j4=i1; if(j5<=0) j5=i4; if(j6<=0) j6=i1;
                    if(j7<=0) j7=i1; if(j8<=0) j8=i4;
                    break;
            case 1010: i1=i2 ; i3=i4 ;
                    if(j1<=0) j1=i4; if(j2<=0) j2=__min(i2,i4); if(j3<=0) j3=i4;
                    if(j4<=0) j4=i2; if(j5<=0) j5=__min(i2,i4); if(j6<=0) j6=i2;
                    if(j7<=0) j7=i2; if(j8<=0) j8=i4;
                    break;
        }
    }
}

```



```

case 1011: i3=__min(i1,i4) ;
            if(j1<=0) j1=i1; if(j2<=0) j2=__min(i2,i4); if(j3<=0) j3=i4;
            if(j4<=0) j4=__min(i1,i2); if(j5<=0) j5=__min(i2,i4); if(j6<=0) j6=i1;
            if(j7<=0) j7=__min(i1,i2); if(j8<=0) j8=i4;
            break;
case 1100: i1=i3 ; i2=i4 ;
            if(j1<=0) j1=i3; if(j2<=0) j2=i4; if(j3<=0) j3=__min(i3,i4);
            if(j4<=0) j4=i4; if(j5<=0) j5=i4; if(j6<=0) j6=i3;
            if(j7<=0) j7=i3; if(j8<=0) j8=__min(i3,i4);
            break;
case 1101: i2=__min(i1,i4) ;
            if(j1<=0) j1=__min(i1,i3); if(j2<=0) j2=i4; if(j3<=0) j3=__min(i3,i4);
            if(j4<=0) j4=i1; if(j5<=0) j5=i4; if(j6<=0) j6=__min(i1,i3);
            if(j7<=0) j7=i1; if(j8<=0) j8=__min(i3,i4);
            break;
case 1110: i1=__min(i2,i3) ;
            if(j1<=0) j1=i3; if(j2<=0) j2=__min(i2,i4); if(j3<=0) j3=__min(i3,i4);
            if(j4<=0) j4=i2; if(j5<=0) j5=__min(i2,i4); if(j6<=0) j6=i3;
            if(j7<=0) j7=i2; if(j8<=0) j8=__min(i3,i4);
            break;
default : return;
        } /* end of switch */

a00=4*i1*r_DIM3 ; a10=(-3*i1+5*i2-j7-j4)*r_DIM2 ; a01=(-3*i1+5*i3-j1-j6)*r_DIM2 ;
a02=(-i1-i3+j1+j6)*r_DIM ; a20=(-i1-i2+j4+j7)*r_DIM ;
a21=i1+i2-i3-i4+j3-j4-j7+j8 ; a12=i1-i2+i3-i4-j1+j2+j5-j6 ;
a11=(2*i1-4*i2-4*i3+6*i4+j1-j2-j3+j4-j5+j6+j7-j8)*r_DIM ;
pas=a02+a01 ; kpas=2*a12 ; lpas=a21+a12+a11 ;
debut=a00 ; kdebut=2*a20 ; ldebut=a10+a20 ;
lpasy=2*(a12+a02) ;

for (x=0,xx=ix;x<r_DIM;x++,xx++) {
    pasy=pas ; in=debut;
    for (y=0,yy=r_HI-iy;y<r_DIM;y++,yy--) {
        if (IS[x][y]>0)
        {
            int pix=in/BD;
            r_Screen->SetPixel(xx,yy,pix);
        }
        in+=pasy;
        pasy+=lpasy;
    }
    pas+=kpas*x+lpas;
    debut+=kdebut*x+ldebut;
}

return; } /* end of mask='*' */

else {
    if(j1<=0) j1=__min(i1,i3); if(j2<=0) j2=__min(i2,i4); if(j3<=0) j3=__min(i3,i4);
    if(j4<=0) j4=__min(i1,i2); if(j5<=0) j5=__min(i2,i4); if(j6<=0) j6=__min(i3,i1);
    if(j7<=0) j7=__min(i1,i2); if(j8<=0) j8=__min(i3,i4);
    a00=4*i1*r_DIM3 ; a10=(-3*i1+5*i2-j7-j4)*r_DIM2 ; a01=(-3*i1+5*i3-j1-j6)*r_DIM2 ;
    a02=(-i1-i3+j1+j6)*r_DIM ; a20=(-i1-i2+j4+j7)*r_DIM ;
    a21=i1+i2-i3-i4+j3-j4-j7+j8 ; a12=i1-i2+i3-i4-j1+j2+j5-j6 ;
    a11=(2*i1-4*i2-4*i3+6*i4+j1-j2-j3+j4-j5+j6+j7-j8)*r_DIM ;
    pas=a02+a01 ; kpas=2*a12 ; lpas=a21+a12+a11 ;
    debut=a00 ; kdebut=2*a20 ; ldebut=a10+a20 ;
    lpasy=2*(a12+a02) ;

    int imax=4*r_DIM3*_max(_max(i1,i2),_max(i3,i4))/BD;
    int imin=4*r_DIM3*_min(_min(i1,i2),_min(i3,i4))/BD;
    for (x=0,xx=ix;x<r_DIM;x++,xx++) {
        pasy=pas ; in=debut;
        for (y=0,yy=r_HI-iy;y<r_DIM;y++,yy--) {
            int pix=in/BD;
            if(pix>imax) pix=imax; else if(pix<imin) pix=imin;
            r_Screen->SetPixel(xx,yy,pix);
        }
        in+=pasy;
        pasy+=lpasy;
    }
    pas+=kpas*x+lpas;
    debut+=kdebut*x+ldebut;
}

return;
} /* end of case mask != '*' */

```

```

}

}

/*****
 *
 *   Set voxel scaling
 *
 *****/
bool RayTracer::SetVolumeScales(double sx, double sy, double sz)
{
    double smin=__min(sx,__min(sy,sz));
    double smax=__max(sx,__max(sy,sz));
    if(smin<=0)
    {
        sprintf(r_error, "Non-positive volume scale");
        return false;
    }
    const int maxscale=64;
    double coef=maxscale/smax;
    r_ALscale[1]=__max(1,(int)__min(maxscale,sx*coef+0.5));
    r_ALscale[2]=__max(1,(int)__min(maxscale,sy*coef+0.5));
    r_ALscale[3]=__max(1,(int)__min(maxscale,sz*coef+0.5));
    return true;
}

/*****
 *
 *   Set observer's eye position
 *
 *****/
void RayTracer::SetEyePosition(double x, double y, double z)
{
    r_Leye[1]=x;
    r_Leye[2]=y;
    r_Leye[3]=z;
}

/*****
 *
 *   Set observer's eye position, where
 *   deg_hor and deg_ver are angles in radians
 *
 *****/
void RayTracer::SetEyePositionOnSphere(double deg_hor, double deg_ver)
{
    double c=cos(deg_ver);
    SetEyePosition(c*cos(deg_hor), c*sin(deg_hor), sin(deg_ver));
}

/*****
 *
 *   Set tracing granularity
 *
 *****/
void RayTracer::SetGranularity(int gr)
{
    r_DIM=gr;
    if(r_DIM>16)        r_DIM=16;
    else if(r_DIM<2)    r_DIM=2;
}

/*****
 *
 *   Set isosurface level to be visualized
 *
 *****/
void RayTracer::SetSurfaceLevel(int lev)    {    r_LEV=lev;    }

/*****
 *
 *   Set max and min body color

```

```

*
*****/
void RayTracer::SetBodyColor(int cmin, int cmax)
{
    cmin=__max(cmin,0);
    if(cmin>cmax)
    {
        int t=cmin; cmin=cmax; cmax=t;
    }
    r_COLb=cmin;           // body minimum
    r_COLf=__max(1,cmax-cmin); // body range: r_COLf=(foreground-r_COLb)
}

/*****
*
*   Return VOXEL value
*
*****/

int RayTracer::F(int x, int y, int z)
{
    /*
    switch(r_RotationCode)
    {
    case 001:  return F_XpYpZm(x,y,z);
    case 010:  return F_XpYmZp(x,y,z);
    case 011:  return F_XpYmZm(x,y,z);
    case 100:  return F_XmYpZp(x,y,z);
    case 101:  return F_XmYpZm(x,y,z);
    case 110:  return F_XmYmZp(x,y,z);
    case 111:  return F_XmYmZm(x,y,z);
    default:   return r_Volume->GetVoxel(x,y,z); //return F_XpYpZp(x,y,z);
    }
    */
    return (this->*F_XYZ)(x,y,z);
}

/*****
*
*   Functions to obtain voxel value for
*   different volume orientation.
*   Array r_N[] must be already modified
*
*****/

int RayTracer::F_XpYpZp(int x, int y, int z)
{
    return r_Volume->GetVoxel(x,y,z);
}
int RayTracer::F_XmYpZp(int x, int y, int z)
{
    return r_Volume->GetVoxel(y,x,r_N[2]-x,z);
}
int RayTracer::F_XpYmZp(int x, int y, int z)
{
    return r_Volume->GetVoxel(r_N[1]-y,x,z);
}
int RayTracer::F_XmYmZp(int x, int y, int z)
{
    return r_Volume->GetVoxel(r_N[1]-x,r_N[2]-y,z);
}
int RayTracer::F_XpYpZm(int x, int y, int z)
{
    return r_Volume->GetVoxel(y,x,r_N[3]-z);
}
int RayTracer::F_XmYpZm(int x, int y, int z)
{
    return r_Volume->GetVoxel(r_N[1]-x,y,r_N[3]-z);
}
int RayTracer::F_XpYmZm(int x, int y, int z)
{
    return r_Volume->GetVoxel(x,r_N[2]-y,r_N[3]-z);
}
int RayTracer::F_XmYmZm(int x, int y, int z)
{
    return r_Volume->GetVoxel(r_N[1]-y,r_N[2]-x,r_N[3]-z);
}

```

	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415	2416	2417	2418	2419	2420	2421	2422	2
--	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	---

[illegible]



```
fmin=m_Fmin;    fmax=m_Fmax;
```

```
// Volume.h: interface for the Volume class.
```

```
//
```

```
////////////////////////////////////
```

```
#if !defined(AFX_VOLUME_H__1FBBBE11_9D18_11D3_8140_000000000000__INCLUDED_)
```

```
#define AFX_VOLUME_H__1FBBBE11_9D18_11D3_8140_000000000000__INCLUDED_
```

```
#if _MSC_VER > 1000
```

```
#pragma once
```

```
#endif // _MSC_VER > 1000
```

```
class Volume
```

```
{
```

```
public:
```

```
void GetMinMax(int& fmin, int& fmax);
```

```
virtual int GetVoxel(int x, int y, int z);
```

```
virtual int GetZSize();
```

```
virtual int GetYSize();
```

```
virtual int GetXSize();
```

```
Volume();
```

```
virtual ~Volume();
```

```
private:
```

```
void FindMinMax();
```

```
int m_Fmin, m_Fmax;
```

```
};
```

```
#endif // !defined(AFX_VOLUME_H__1FBBBE11_9D18_11D3_8140_000000000000__INCLUDED_)
```





```

    }
    return (p>>m_PixShift)&m_PixMask;
};
inline long RawPixelDecoder::GetBufferedPixel(UINT32 n)
{
    if(m_SamplesPerPixel==1)
    {
        return GetBufferedPixelSample(n);
    }
    else
    {
        n *= m_SamplesPerPixel;
        long p=GetBufferedPixelSample(n);
        for(int i=1; i<m_SamplesPerPixel; i++)
        {
            n++;
            p += GetBufferedPixelSample(n);
        }
        p /= m_SamplesPerPixel;
        return p;
    }
}

```

```

/////////////////////////////////////////////////////////////////
//
// RawPixelEncoder class.
//
/////////////////////////////////////////////////////////////////

```

```

class RawPixelEncoder

```

```

{
public:

```

```

    void      TransferDataToVR(VR *vr);
    bool      SetSize(UINT32 size);
    UINT32    AddData(BYTE* buf, UINT32 buf_size,
                     UINT32 fliprawbytes=0);

```

```

    RawPixelEncoder();
    virtual ~RawPixelEncoder();

```

```

private:

```

```

    bool      m_ReleaseBufferMemory;
    BYTE*     m_pBuffer;
    UINT32    m_BufferPtr;
    UINT32    m_Size;

    void      ReleaseBuffer();

```

```

#endif // !defined(AFX_PIXELS_H__2A361EE3_CEAB_11D3_AF48_000000000000__INCLUDED_)

```

```

// pixels.cpp
//
//
#include "pixels.h"

// Construction/Destruction
//
RawPixelDecoder::RawPixelDecoder(UINT32 npixels, int bits_alloc,
                                int bits_stored, int high_bit,
                                int samples_per_pixel, int endian/*=1*/)
{
    m_BytesPerPixel=0;
    m_LitEndian=1;
    m_nBytes=m_nPixels=m_nSamples=m_SamplesPerPixel=0;
    m_pBuffer=NULL; m_ReleaseBufferMemory=false;
    m_PixMask=m_PixShift=0;
    Initialize( npixels, bits_alloc, bits_stored, high_bit,
               samples_per_pixel, endian );
}

RawPixelDecoder::~RawPixelDecoder() { ReleaseBuffer(); }

/*****
*
*   Initialize buffer parameters
*
*****/
void RawPixelDecoder::Initialize(UINT32 npixels, int bits_alloc,
                                int bits_stored, int high_bit,
                                int samples_per_pixel, int endian/*=1*/)
{
    m_ReleaseBufferMemory=false;
    m_nPixels=npixels; // number of pixels to read; one pixel may have several samples
    m_BytesPerPixel=(bits_alloc+7)/8; // number of bytes per pixel
    m_SamplesPerPixel=samples_per_pixel; // samples per pixel
    m_nSamples=m_nPixels*m_SamplesPerPixel;
    m_LitEndian=endian; // endian type
    m_nBytes=m_nSamples*m_BytesPerPixel; // total number of bytes
    if(!SetPixelMask(bits_alloc, bits_stored, high_bit)) m_BytesPerPixel=0;
}

/*****
*
*   Validate current parameter values
*
*****/
bool RawPixelDecoder::Valid()
{
    if( m_BytesPerPixel<=0 || m_BytesPerPixel>4 ||
        m_SamplesPerPixel<=0 || m_SamplesPerPixel>4 ||
        m_nPixels<=4 || m_nSamples<=4 || m_nBytes<=4 )
    {
        return false;
    }
    else return true;
}

/*****
*
*   Compute pixel mask parameters
*
*****/
bool RawPixelDecoder::SetPixelMask(int bits_allocated, int bits_stored, int high_bit)
{
    if( bits_allocated<bits_stored || bits_allocated<high_bit ||
        bits_allocated<=0 || bits_stored<=0 ) return false;

    m_PixMask=1;
    for(int i=1; i<bits_stored; i++)
    {
        m_PixMask = (m_PixMask<<1)+1;
    }
    m_PixShift=high_bit-bits_stored+1;
    return true;
}

/*****
*
*   Load pixel data, in bytes, from the input file or buffer
*
*****/

```

```

*
*****/
bool RawPixelDecoder::Load_Pixels(FILE * infile)
{
    if(!Valid())    return false;
    ReleaseBuffer();
    try
    {
        m_pBuffer=new BYTE[m_nBytes];    // pixel buffer
        if(!m_pBuffer) return false; // out of memory
    }
    catch(...) {    return false;    }
    memset(m_pBuffer,0,m_nBytes);
    m_ReleaseBufferMemory=true;
    if(fread(m_pBuffer,1,m_nBytes, infile) < m_nBytes) return false;
    Digest();
    return true;
}

bool RawPixelDecoder::Load_Pixels(BYTE *data, UINT32 data_size)
{
    if(!Valid())    return false;
    ReleaseBuffer();
    if(m_nBytes>data_size) return false;    // incomplete data
    m_ReleaseBufferMemory=false;
    m_pBuffer=data;
    Digest();
    return true;
}

}

/*****
*
* Process loaded pixel bytes. Return pixel range
*
*****/
void RawPixelDecoder::Digest()
{
    UINT32 i;

    // If not little endian - go swap bytes
    if(!m_LitEndian)
    {
        ::SwitchEndian(m_pBuffer, m_nBytes, m_BytesPerPixel);
    }

    // Find min and max pixel SAMPLE values
    long p;
    m_PixMin=GetBufferedPixelSample(0);
    m_PixMax=m_PixMin+1;
    for(i=1; i<m_nSamples; i++)
    {
        p=GetBufferedPixelSample(i);
        if(p>m_PixMax) m_PixMax=p;
        else if(p<m_PixMin) m_PixMin=p;
    }
}

/*****
*
* Release allocated buffer memory
*
*****/
void RawPixelDecoder::ReleaseBuffer()
{
    if(m_ReleaseBufferMemory)
    {
        if(m_pBuffer)
        {
            delete [] m_pBuffer;
            m_pBuffer=NULL;
        }
        m_ReleaseBufferMemory=false;
    }
}

```

```

int nXpos = 0;
int nYpos1 = nTop + 1;
int nYpos2 = nBottom - 2;

for (int i = 0; i < nNumBands; i++)
{
    nXpos = nAdjust + (i * IND_BAND_WIDTH);

    pDC->SelectObject(&m_penColorDarker);
    pDC->MoveTo(nXpos + 1, nTop);
    pDC->LineTo(nXpos + nHeight, nBottom);

    pDC->SelectObject(&m_penColorDark);
    pDC->MoveTo(nXpos + 2, nTop);
    pDC->LineTo(nXpos + nHeight + 1, nBottom);
    pDC->MoveTo(nXpos + 10, nTop);
    pDC->LineTo(nXpos + nHeight + 9, nBottom);

    pDC->SelectObject(&m_penColor);
    pDC->MoveTo(nXpos + 3, nTop);
    pDC->LineTo(nXpos + nHeight + 2, nBottom);
    pDC->MoveTo(nXpos + 9, nTop);
    pDC->LineTo(nXpos + nHeight + 8, nBottom);

    pDC->SelectObject(&m_penColorLight);
    pDC->MoveTo(nXpos + 4, nTop);
    pDC->LineTo(nXpos + nHeight + 3, nBottom);
    pDC->MoveTo(nXpos + 8, nTop);
    pDC->LineTo(nXpos + nHeight + 7, nBottom);

    pDC->SelectObject(&m_penColorLighter);
    pDC->MoveTo(nXpos + 5, nTop);
    pDC->LineTo(nXpos + nHeight + 4, nBottom);
    pDC->MoveTo(nXpos + 7, nTop);
    pDC->LineTo(nXpos + nHeight + 6, nBottom);
} // for the number of bands
} // if indeterminate
else
{
    int nRight = rect.right;

    pDC->MoveTo(nLeft + 2, nBottom - 4);
    pDC->LineTo(nRight - 2, nBottom - 4);
    pDC->MoveTo(nLeft + 2, nTop + 2);
    pDC->LineTo(nRight - 2, nTop + 2);
    pDC->SetPixel(nLeft + 1, nBottom - 3, m_crColorLight);
    pDC->SetPixel(nLeft + 1, nTop + 1, m_crColorLight);

    pDC->SelectObject(&m_penColorLighter);
    pDC->MoveTo(nLeft + 2, nBottom - 5);
    pDC->LineTo(nRight - 3, nBottom - 5);
    pDC->LineTo(nRight - 3, nTop + 3);
    pDC->LineTo(nLeft + 1, nTop + 3);
    pDC->SetPixel(nLeft + 1, nBottom - 4, m_crColorLighter);
    pDC->SetPixel(nLeft + 1, nTop + 2, m_crColorLighter);

    pDC->SelectObject(&m_penColor);
    pDC->MoveTo(nLeft, nBottom - 1);
    pDC->LineTo(nLeft, nTop);
    pDC->LineTo(nLeft + 2, nTop);
    pDC->SetPixel(nLeft + 1, nBottom - 2, m_crColor);
    pDC->MoveTo(nLeft + 2, nBottom - 3);
    pDC->LineTo(nRight - 2, nBottom - 3);
    pDC->MoveTo(nLeft + 2, nTop + 1);
    pDC->LineTo(nRight - 1, nTop + 1);

    pDC->SelectObject(&m_penColorDark);
    pDC->MoveTo(nLeft + 2, nBottom - 2);
    pDC->LineTo(nRight - 2, nBottom - 2);
    pDC->LineTo(nRight - 2, nTop + 1);
    pDC->MoveTo(nLeft + 2, nTop);
    pDC->LineTo(nRight, nTop);
    pDC->SetPixel(nLeft + 1, nBottom - 1, m_crColorDark);

```

```

pDC->SelectObject(&m_penColorDarker);
pDC->MoveTo(nLeft + 2, nBottom - 1);
pDC->LineTo(nRight - 1, nBottom - 1);
pDC->LineTo(nRight - 1, nTop);

pDC->SelectObject(&m_penShadow);
pDC->MoveTo(nRight, nTop);
pDC->LineTo(nRight, nBottom);

pDC->SelectObject(&m_penLiteShadow);
pDC->MoveTo(nRight + 1, nTop);
pDC->LineTo(nRight + 1, nBottom);
} // if not indeterminate

pDC->SelectObject(pOldPen);
} // DrawHorizontalBar

//-----
//
void CMacProgressCtrl::DrawVerticalBar(CDC *pDC, const CRect rect)
//
// Return Value:      None.
//
// Parameters       :   pDC - Specifies the device context object.
//                    rect - Specifies the rectangle of the progress bar.
//
// Remarks          :   Draws a vertical progress bar.
//
{
    int nHeight = rect.Height();
    if (!nHeight)
        return;

    int nLeft = rect.left;
    int nTop = rect.top;
    int nRight = rect.right;
    int nBottom = rect.bottom;

    CPen *pOldPen = pDC->SelectObject(&m_penColor);

    if (m_bIndeterminate)
    {
        int nNumBands = (nHeight / IND_BAND_WIDTH) + 2;
        int nHeight = rect.Width() + 1;

        int nAdjust = nBottom - m_nIndOffset;
        int nXpos1 = nLeft;
        int nXpos2 = nRight + 1;
        int nYpos = nTop + 1;

        for (int i = 0; i < nNumBands; i++)
        {
            nYpos = nAdjust - (i * IND_BAND_WIDTH);

            pDC->SelectObject(&m_penColorDarker);
            pDC->MoveTo(nXpos1, nYpos);
            pDC->LineTo(nXpos2, nYpos + nHeight);

            pDC->SelectObject(&m_penColorDark);
            pDC->MoveTo(nXpos1, nYpos + 1);
            pDC->LineTo(nXpos2, nYpos + nHeight + 1);
            pDC->MoveTo(nXpos1, nYpos + 9);
            pDC->LineTo(nXpos2, nYpos + nHeight + 9);

            pDC->SelectObject(&m_penColor);
            pDC->MoveTo(nXpos1, nYpos + 2);
            pDC->LineTo(nXpos2, nYpos + nHeight + 2);
            pDC->MoveTo(nXpos1, nYpos + 8);
            pDC->LineTo(nXpos2, nYpos + nHeight + 8);

            pDC->SelectObject(&m_penColorLight);
            pDC->MoveTo(nXpos1, nYpos + 3);
            pDC->LineTo(nXpos2, nYpos + nHeight + 3);
            pDC->MoveTo(nXpos1, nYpos + 7);

```

```

        pDC->LineTo(nXpos2, nYpos + nHeight + 7);

        pDC->SelectObject(&m_penColorLighter);
        pDC->MoveTo(nXpos1, nYpos + 4);
        pDC->LineTo(nXpos2, nYpos + nHeight + 4);
        pDC->MoveTo(nXpos1, nYpos + 6);
        pDC->LineTo(nXpos2, nYpos + nHeight + 6);
    } // for the number of bands
} // if indeterminate
else
{
    if (nHeight > 3)
    {
        pDC->MoveTo(nLeft, nTop + 1);
        pDC->LineTo(nLeft, nTop);
        pDC->LineTo(nRight, nTop);
        pDC->MoveTo(nLeft + 1, nBottom - 2);
        pDC->LineTo(nLeft + 1, nTop + 1);
        pDC->MoveTo(nRight - 3, nBottom - 3);
        pDC->LineTo(nRight - 3, nTop + 1);
        pDC->SetPixel(nRight - 2, nTop + 1, m_crColor);

        pDC->SelectObject(&m_penColorLight);
        pDC->MoveTo(nLeft + 2, nBottom - 3);
        pDC->LineTo(nLeft + 2, nTop + 1);
        pDC->MoveTo(nRight - 4, nBottom - 3);
        pDC->LineTo(nRight - 4, nTop + 1);
        pDC->SetPixel(nLeft + 1, nTop + 1, m_crColorLight);
        pDC->SetPixel(nRight - 3, nTop + 1, m_crColorLight);

        pDC->SelectObject(&m_penColorLighter);
        pDC->MoveTo(nLeft + 3, nBottom - 3);
        pDC->LineTo(nLeft + 3, nTop + 1);
        pDC->MoveTo(nRight - 5, nBottom - 3);
        pDC->LineTo(nRight - 5, nTop + 1);
        pDC->SetPixel(nLeft + 2, nTop + 1, m_crColorLighter);
        pDC->SetPixel(nRight - 4, nTop + 1, m_crColorLighter);

        pDC->SelectObject(&m_penColorDark);
        pDC->MoveTo(nLeft, nBottom - 1);
        pDC->LineTo(nLeft, nTop + 1);
        pDC->MoveTo(nLeft + 2, nBottom - 2);
        pDC->LineTo(nRight - 2, nBottom - 2);
        pDC->LineTo(nRight - 2, nTop + 1);
        pDC->SetPixel(nRight - 1, nTop + 1, m_crColorDark);

        pDC->SelectObject(&m_penColorDarker);
        pDC->MoveTo(nLeft + 1, nBottom - 1);
        pDC->LineTo(nRight - 1, nBottom - 1);
        pDC->LineTo(nRight - 1, nTop + 1);
    }
}
else
{
    CBrush br(m_crColor);
    CBrush *pOldBrush = pDC->SelectObject(&br);
    pDC->SelectObject(&m_penColorDark);
    pDC->Rectangle(rect);
    pDC->SelectObject(pOldBrush);
}
} // if not indeterminate

pDC->SelectObject(pOldPen);
} // DrawVerticalBar

//-----
//
BOOL CMacProgressCtrl::OnEraseBkgnd(CDC* pDC)
//
// Return Value:    Nonzero if it erases the background; otherwise 0.
//
// Parameters      :    pDC - Specifies the device-context object.
//
// Remarks         :    The framework calls this member function when the
//                      CWnd object background needs erasing (for example,

```

```

// when resized). It is called to prepare an invalidated
// region for painting.
//
{
    return TRUE;
} // OnEraseBkgnd

//-----
//
void CMacProgressCtrl::GetColors()
//
// Return Value:    None.
//
// Parameters      :    None.
//
// Remarks         :    Calculates the lighter and darker colors, as well as
//                      the shadow colors.
//
{
    m_crColorLight = LightenColor(m_crColor, 51);
    m_crColorLighter = LightenColor(m_crColorLight, 51);
    m_crColorLightest = LightenColor(m_crColorLighter, 51);
    m_crColorDark = DarkenColor(m_crColor, 51);
    m_crColorDarker = DarkenColor(m_crColorDark, 51);
    m_crDkShadow = ::GetSysColor(COLOR_3DDKSHADOW);
    m_crLiteShadow = ::GetSysColor(COLOR_3DSHADOW);

    // Get a color halfway between COLOR_3DDKSHADOW and COLOR_3DSHADOW
    BYTE byRed3DDkShadow = GetRValue(m_crDkShadow);
    BYTE byRed3DLiteShadow = GetRValue(m_crLiteShadow);
    BYTE byGreen3DDkShadow = GetGValue(m_crDkShadow);
    BYTE byGreen3DLiteShadow = GetGValue(m_crLiteShadow);
    BYTE byBlue3DDkShadow = GetBValue(m_crDkShadow);
    BYTE byBlue3DLiteShadow = GetBValue(m_crLiteShadow);

    m_crShadow = RGB(byRed3DLiteShadow + ((byRed3DDkShadow - byRed3DLiteShadow) >> 1),
                    byGreen3DLiteShadow + ((byGreen3DDkShadow - byGreen3DLiteShadow) >> 1),
                    byBlue3DLiteShadow + ((byBlue3DDkShadow - byBlue3DLiteShadow) >> 1));
} // GetColors

//-----
//
void CMacProgressCtrl::SetColor(COLORREF crColor)
//
// Return Value:    None.
//
// Parameters      :    crColor - New color.
//
// Remarks         :    Sets the progress bar control's color. The lighter
//                      darker colors are recalculated, and the pens recreated.
//
{
    m_crColor = crColor;
    GetColors();
    CreatePens();
    RedrawWindow();
} // SetColor

//-----
//
COLORREF CMacProgressCtrl::GetColor()
//
// Return Value:    The current color.
//
// Parameters      :    None.
//
// Remarks         :    Returns the progress bar control's current color.
//
{
    return m_crColor;
} // GetColor

//-----
//

```

```

void CMacProgressCtrl::CreatePens()
//
// Return Value:    None.
//
// Parameters      :    None.
//
// Remarks         :    Deletes the pen objects, if necessary, and creates them.
//
{
    DeletePens();

    m_penColorLight.CreatePen(PS_SOLID, 1, m_crColorLight);
    m_penColorLighter.CreatePen(PS_SOLID, 1, m_crColorLighter);
    m_penColor.CreatePen(PS_SOLID, 1, m_crColor);
    m_penColorDark.CreatePen(PS_SOLID, 1, m_crColorDark);
    m_penColorDarker.CreatePen(PS_SOLID, 1, m_crColorDarker);
    m_penDkShadow.CreatePen(PS_SOLID, 1, m_crDkShadow);
    m_penShadow.CreatePen(PS_SOLID, 1, m_crShadow);
    m_penLiteShadow.CreatePen(PS_SOLID, 1, m_crLiteShadow);
} // CreatePens

```

```

//-----
//
void CMacProgressCtrl::DeletePens()
//
// Return Value:    None.
//
// Parameters      :    None.
//
// Remarks         :    Deletes the pen objects.
//
{
    if (m_penColorLight.m_hObject)
        m_penColorLight.DeleteObject();
    if (m_penColorLighter.m_hObject)
        m_penColorLighter.DeleteObject();
    if (m_penColor.m_hObject)
        m_penColor.DeleteObject();
    if (m_penColorDark.m_hObject)
        m_penColorDark.DeleteObject();
    if (m_penColorDarker.m_hObject)
        m_penColorDarker.DeleteObject();
    if (m_penDkShadow.m_hObject)
        m_penDkShadow.DeleteObject();
    if (m_penShadow.m_hObject)
        m_penShadow.DeleteObject();
    if (m_penLiteShadow.m_hObject)
        m_penLiteShadow.DeleteObject();
} // DeletePens

```

```

//-----
//
void CMacProgressCtrl::SetIndeterminate(BOOL bIndeterminate)
//
// Return Value:    None.
//
// Parameters      :    bIndeterminate - Specifies the indeterminate state.
//
// Remarks         :    Sets the indeterminate flag.
//
{
    m_bIndeterminate = bIndeterminate;

    if (m_bIndeterminate)
    {
        CRect rect;
        GetClientRect(rect);
        m_nIndOffset = 0;

        RedrawWindow();
        SetTimer(IDT_INDETERMINATE, 25, NULL);
    }
    else
    {

```



```

KillTimer(IDT_INDETERMINATE);
RedrawWindow();
}
} // SetIndeterminate

```

```

//-----
//
BOOL CMacProgressCtrl::GetIndeterminate()

```

```

//
// Return Value:    m_bIndeterminate.
//
// Parameters      :    None.
//
// Remarks         :    Returns m_bIndeterminate.
//
//
{
    return m_bIndeterminate;
} // GetIndeterminate

```

```

//-----
//
void CMacProgressCtrl::OnTimer(UINT nIDEvent)

```

```

//
// Return Value:    None.
//
// Parameters      :    nIDEvent - Specifies the identifier of the timer.
//
// Remarks         :    The framework calls this member function after each
//                      interval specified in the SetTimer member function used
//                      to install a timer.

```

```

{
    // Increment the indeterminate bar offset and redraw the window.
    if (nIDEvent == IDT_INDETERMINATE)
    {
        KillTimer(nIDEvent);

        if (++m_nIndOffset > IND_BAND_WIDTH - 1)
            m_nIndOffset = 0;
        RedrawWindow();

        SetTimer(IDT_INDETERMINATE, 25, NULL);
    }
} // OnTimer

```

```

//=====
// OProgress Class
//=====
//=====
// Construction/Destruction
//=====

```

```

OProgress::OProgress()
{
}

```

```

OProgress::~OProgress()
{
}

```

```

/*****
*
* Initialize progress control
* and insert it in the main frame window status bar
*
*****/

```

```

bool OProgress::Initialize()
{
    CMDIFrameWnd* mf=(CMDIFrameWnd*)AfxGetMainWnd();
    if(!mf)
    {
        AfxMessageBox("Failed to create progress control");
    }
}

```

```

    return false;
}
CStatusBar* main_status_bar=(CStatusBar*)(mf->GetMessageBar());
if(!main_status_bar)
{
    AfxMessageBox("Failed to create progress control");
    return false;
}
RECT rc;    main_status_bar->GetItemRect(1,&rc);
if(!Create(WS_CHILD | WS_VISIBLE, rc, main_status_bar, 1))
{
    AfxMessageBox("Failed to create progress control");
    return false;
}
SetRange(0,100);    SetPos(0);    SetColor(RGB(0,10,0));
return true;
}

```

```

/*****
*
*   Show progress with optional message string
*   in the main frame window status bar
*
*****/
void OProgress::ShowProgress(int percent, char *info /*=NULL*/)
{
    if(GetSafeHwnd()==NULL) return;
    if(percent<=0)    percent=0;
    else if(percent>100)    percent=100;
    SetPos(percent);
    if(percent==0 || percent==100)    info="Ready";
    if(info)
    {
        CMDIFrameWnd* mf=(CMDIFrameWnd*)AfxGetMainWnd();
        if(!mf) return;
        CStatusBar* main_status_bar=(CStatusBar*)(mf->GetMessageBar());
        if(!main_status_bar)    return;
        main_status_bar->SetPaneText(0,info);
    }
}

```

```

// IEToolBar.h: interface for the IEToolBar class.
//
/////////////////////////////////////////////////////////////////
#if !defined(AFX_IEToolBar_H_467C453D_E943_11D3_977E_00105A21774F__INCLUDED_)
#define AFX_IEToolBar_H_467C453D_E943_11D3_977E_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include <afxext.h>

class IEToolBar : public CToolBar
{
public:
    void        SetInsertedControlText(CString s);
    bool        TrackDropDownMenu(UINT buttonID, CWnd* pParent);
    bool        AttachDropDown(UINT buttonID, UINT menuID, UINT menuItemID);
    bool        MakeCStatic(UINT id);
    BOOL        CreateIE(    CWnd* pParentWnd, UINT img_width,
                            UINT img_height, UINT resource );

    IEToolBar();
    ~IEToolBar();

private:
    UINT        m_ImageWidth, m_ImageHeight;
    UINT*       menuIDs;
    UINT*       menuItemIDs;
    CStatic*    m_Static;

    void        SetButtonsIE();
    bool        InsertControl(int ctrl_type, UINT nID, CString title="");
    CMenu*      GetSubmenuFromID(CMenu* menu, UINT id);
}

#endif // !defined(AFX_IEToolBar_H_467C453D_E943_11D3_977E_00105A21774F__INCLUDED_)

```

```
// IEToolBar.cpp: implementation of the IEToolBar class.
//
//
//
#include "stdafx.h"
#include "IEToolBar.h"

//
// Construction/Destruction
//
IEToolBar::IEToolBar()
{
    m_Static=NULL;
    menuIDs=NULL;
    menuItemIDs=NULL;
}

IEToolBar::~IEToolBar()
{
    if(m_Static) delete m_Static;
    if(menuIDs) delete [] menuIDs;
    if(menuItemIDs) delete [] menuItemIDs;
}

/*****
*
* Creating toolbars with IE style
*
*****/
BOOL IEToolBar::CreateIE(CWnd *pParentWnd, UINT img_width,
                        UINT img_height, UINT resource)
{
    m_Static=NULL; menuIDs=NULL; menuItemIDs=NULL;
    m_ImageWidth=img_width; m_ImageHeight=img_height;
    if(CToolBar::CreateEx(pParentWnd,TBSTYLE_FLAT,
        WS_CHILD | WS_VISIBLE | CBRS_ALIGN_TOP |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC)==FALSE)
    {
        AfxMessageBox("Failed to create toolbar");
        return FALSE;
    }
    if(LoadToolBar(resource)==FALSE)
    {
        AfxMessageBox("Failed to load toolbar");
        return FALSE;
    }
    SetButtonsIE();

    // TODO: Remove this if you don't want tool tips or a resizable toolbar
    SetBarStyle(GetBarStyle() | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);
    // Support for dropdown arrows
    GetToolBarCtrl().SetExtendedStyle(TBSTYLE_EX_DRAWDDARROWS);
    menuIDs = new UINT[GetCount()+5];
    menuItemIDs = new UINT[GetCount()+5];
    for(int i = 0; i < GetCount()+5; i++) menuItemIDs[i]=menuIDs[i]=0;

    return TRUE;
}

/*****
*
* Set IE-like flat buttons with text
*
*****/
void IEToolBar::SetButtonsIE()
{
    // Add text to each button
    int tlength=5;
    for(int i = 0; i < GetCount(); i++)
    {
        UINT id = GetItemID(i);
        CString s; if(!s.LoadString(id)) continue;
        int j = s.Find('\n');
        if(j < 0) continue;
        else s=s.Mid(j+1);
    }
}

```

```

        if(s.GetLength()>tlength)    s=s.Left(tlength);
        SetButtonText(i,s);
    }
    // Resize buttons to include text
    CRect rect;
    GetItemRect(0,&rect);
    SetSizes(rect.Size(),CSize(m_ImageWidth,m_ImageHeight));
}
/*****
*
*   Associate given id with CStatic control
*
*****/
bool IEToolBar::MakeCStatic(UINT id)
{
    return InsertControl(1,id,"0");
}
/*****
*
*   Insert a control instead nID button
*
*****/
bool IEToolBar::InsertControl(int ctrl_type, UINT nID,
                              CString title /*="" */)
{
    DWORD dwStyle = WS_CHILD | WS_VISIBLE;
    CWnd* pCtrl=NULL;
    CRect rect;

    // Make sure the id is valid
    int index = CommandToIndex( nID );  if(index<0) return false;
    GetItemRect(index,rect); rect.left += 15;
    SetButtonInfo(index, nID, TBBS_SEPARATOR, rect.Width());

    // Insert the control
    switch(ctrl_type)
    {
    case 1: // CStatic
        if(m_Static) return false;
        m_Static = new CStatic();  if(!m_Static) return false;
        dwStyle |= SS_CENTER;
        if(!m_Static->Create(title, dwStyle, rect, this, nID))
        {
            delete m_Static; m_Static=NULL; return false;
        }
        pCtrl=m_Static;
        break;
    default: return false;
    }
    GetItemRect(index, &rect );
    pCtrl->SetWindowPos(0, rect.left, rect.top, 0, 0,
        SWP_NOZORDER | SWP_NOACTIVATE | SWP_NOSIZE | SWP_NOCOPYBITS );
    pCtrl->ShowWindow( SW_SHOW );
    return true;
}
/*****
*
*   If a control was inserted into the toolbar
*   set it text to a given string
*
*****/
void IEToolBar::SetInsertedControlText(CString s)
{
    if(m_Static)    m_Static->SetWindowText(CString("\n")+s);
}
/*****
*
*   Attach dropdwln arrow and menu to a given button
*
*****/

```

```

bool IEToolBar::AttachDropDown(UINT buttonID, UINT menuID, UINT menuItemID)
{
    // Make sure the id is valid
    int index = CommandToIndex(buttonID);
    if(index<0 || index>=GetCount()) return false;
    DWORD dwStyle = GetButtonStyle(index);
    dwStyle |= TBSTYLE_DROPDOWN;
    SetButtonStyle(index, dwStyle);
    menuIDs[index]=menuID;
    menuItemIDs[index]=menuItemID;
    return true;
}

/*****
*
*   Display dropdown menu
*
*****/
bool IEToolBar::TrackDropDownMenu(UINT buttonID, CWnd *pParent)
{
    // Make sure the id is valid
    int index = CommandToIndex(buttonID);
    if(index<0 || index>=GetCount()) return false;
    // Find menu ID
    UINT menuID=menuIDs[index];
    if(menuID==0) return true; // no menu attached
    // Load and display popup menu
    CMenu menu; menu.LoadMenu(menuID);
    CMenu* pPopup=GetSubMenuFromID(&menu,menuItemIDs[index]);

    if(!pPopup) return true; // no such submenu
    CRect rc;
    this->SendMessage(TB_GETRECT, buttonID, (LPARAM)&rc);
    this->ClientToScreen(&rc);
    pPopup->TrackPopupMenu( TPM_LEFTALIGN | TPM_LEFTBUTTON | TPM_VERTICAL,
        rc.left, rc.bottom, pParent, &rc);
    return true;
}

/*****
*
*   Find submenu which contains given menu id
*
*****/
CMenu* IEToolBar::GetSubMenuFromID(CMenu* menu, UINT id)
{
    CMenu* sub;
    if(!menu) return NULL;
    UINT c = menu->GetMenuItemCount();
    if(c<=0) return NULL;
    for(UINT i=0; i<c; i++)
    {
        if(menu->GetMenuItemID(i)==id) return menu;
        sub = menu->GetSubMenu(i);
        if(!sub) continue;
        sub=GetSubMenuFromID(sub,id);
        if(sub) return sub;
    }
    return NULL;
}

```

```

// RayTracer.h: interface for the RayTracer class.
//
/////////////////////////////////////////////////////////////////

#if !defined(AFX_RAYTRACER_H__9741FB4F_8B17_11D3_9720_00105A21774F__INCLUDED_)
#define AFX_RAYTRACER_H__9741FB4F_8B17_11D3_9720_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include "RTScreen.h"
#include "Volume.h"

class RayTracer
{
public:
    void SetBodyColor(int cmin, int cmax);
    void SetSurfaceLevel(int lev);
    void SetGranularity(int gr);
    void SetEyePosition(double x, double y, double z);
    void SetEyePositionOnSphere(double deg_hor, double deg_ver);
    bool SetVolumeScales(double sx, double sy, double sz);
    char r_error[64];

    double DoTracing();
    RayTracer(Volume* v, RTScreen* rts);
    virtual ~RayTracer();

private:
    signed char IS[16][16],AX,AY,AZ,r_ox,r_oy,r_oz;
    int r_ALscale[4],r_N[4],Na[4],NX[14],IED[4],IL[4],P[4],TOP[4];
    int r_COLb,r_COLf,r_COLc,r_COLL,r_LE,r_HI,r_MAXx,r_MAXy,r_DIM,r_DIM1;
    int Fmax,Fmin,r_LEV;
    int r_RotationCode;
    long BD,IL12,IL13,IL23,DL1,DL2,PE,QE,AA,r_DIM2,r_DIM3;
    double r_Leye[4],LL[4],AP[4],B1,B2,FIL;
    double DOWN,CU1,CU2,CV1,CV2,CV3,CF1,CF2,CF3;
    double LLL,D1,D2,Fk;
    RTScreen* r_Screen;
    Volume* r_Volume;

    void EDGE();
    void INST (int p,int q,int u,int v);
    bool PICTURE (char how);
    char TEST (int w1,int w2,int w3,int w4);
    int Q_JUMP ();
    double FUNC(long x,long y,long z);
    double JUMP (double t);

    // inline functions
    inline void CHESS (int uu,int vv,int i1,int i2,int i3,int i4,int j1,int j2,
        int j3,int j4,int j5,int j6,int j7,int j8 );
    inline void GURO2(char mask,long i1,long i2,long i3,long i4,long j1,long j2,
        long j3,long j4,long j5,long j6,long j7,long j8,int ix,int iy);
    inline bool CUBESIGN(int a1,int a2,int a3,int a4,int a5,int a6,int a7,int a8);
    inline int INTENS (double a,double b,double c,double d,double t);
    inline int F(int x, int y, int z);
    inline int F_XpYpZp(int x, int y, int z);
    inline int F_XmYpZp(int x, int y, int z);
    inline int F_XpYmZp(int x, int y, int z);
    inline int F_XmYmZp(int x, int y, int z);
    inline int F_XpYpZm(int x, int y, int z);
    inline int F_XmYpZm(int x, int y, int z);
    inline int F_XpYmZm(int x, int y, int z);
    inline int F_XmYmZm(int x, int y, int z);
};

#endif // !defined(AFX_RAYTRACER_H__9741FB4F_8B17_11D3_9720_00105A21774F__INCLUDED_)

```





```

// RayTracer.cpp: implementation of the RayTracer class.
//
//
#include "RayTracer.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//
// Construction/Destruction
//

RayTracer::RayTracer(Volume* v, RTScreen* rts)
{
    r_Volume=v; r_Screen=rts;
    SetVolumeScales(1,1,1);
    SetEyePosition(1,1,1);
    SetGranularity(4);
    SetSurfaceLevel(0);
    SetBodyColor(10,255);
}

RayTracer::~RayTracer() {}

int (RayTracer::*F_XYZ)(int x, int y, int z);

/*
 * Ray tracing; returns tracing time
 */
double RayTracer::DoTracing()
{
    int i,j,j0,j1,k,m,m1,m2,lef,mid,rig,idim,jdim;
    int jf0,jf1,jl0,jl1,jm0,jm1,jmin,jmax,p0,p1,p2,p3,jj1,jj2,jj5,jj6,wd[4][500];
    double u1,u2,v1,v2,v3,du,un1,un2,vn1,vn2,vn3,a,b,h1,h2,e1,e2,e3,e4;
    double x0,y0,z0,t,ddt,dt[4];
    double x[4],td[3][500];
    clock_t clock_start,clock_end;

    /*
    ***** RAY TRACING PARAMETERS
    //Volume sizes in (x,y,z)
    r_N[1]=r_Volume->GetXSize();
    r_N[2]=r_Volume->GetYSize();
    r_N[3]=r_Volume->GetZSize();
    // int Axes numbers, to account for views from any octant
    r_OX=1; r_OY=2; r_OZ=3;
    // int Colors
    r_COLc=10; // non-body background
    r_COLL=5; // color to draw the volume frame
    r_MAXX=800; r_MAXY=700; // max screen sizes for traced picture
    r_LE=r_MAXX-10; r_HI=r_MAXY-10; // actual screen sizes for traced picture

    ***** Do we need scaling ?
    r_Volume->GetMinMax(Fmin,Fmax);
    PE=Fmax-Fmin;
    if(r_LEV<Fmin || r_LEV>Fmax)
    {
        sprintf(r_error,"Chosen level must be in [%d,%d] interval",Fmin,Fmax);
        return -1.0;
    }
    if(PE>500)
    {
        Fk=(double)(500./PE);
        r_LEV=(int)(r_LEV*Fk+0.5);
        sprintf(r_error,"Data needs to be scaled, current range is [%d,%d]",Fmin,Fmax);
        return -1.0; // Need F(i,j,k) -> (F[i][j][k]-Fmin)*Fk
    }
    */
}

```

```

else      Fk=1.;
FIL=(double)(10000*sqrt( 5./(Fk*(Fmax-Fmin+1)) ));

/***** Compute picture parameters *****/
if(!PICTURE ('Y')) return -1.0;
AX=1;      AY=2;      AZ=3;
u1=CU1;    u2=CU2;
v1=CV1;    v2=CV2;    v3=CV3;
du=AP[1];  un1=AP[2]; un2=AP[3];
vn1=CF1;   vn2=CF2;   vn3=CF3;
m1=r_DIM1; m2=r_DIM2;

/***** Accomodate different quadrants *****/
r_RotationCode=0;
if (r_Leye[1]<0) r_RotationCode += 1;
if (r_Leye[2]<0) r_RotationCode += 10;
if (r_Leye[3]<0) r_RotationCode += 100;
switch(r_RotationCode)
{
case 001:
    F_XYZ=(this->F_XpYpZm);
    break;
case 010:
    F_XYZ=(this->F_XpYmZp);
    break;
case 011:
    F_XYZ=(this->F_XpYmZm);
    break;
case 100:
    F_XYZ=(this->F_XmYpZp);
    break;
case 101:
    F_XYZ=(this->F_XmYpZm);
    break;
case 110:
    F_XYZ=(this->F_XmYmZp);
    break;
case 111:
    F_XYZ=(this->F_XmYmZm);
    break;
default:
    F_XYZ=(this->F_XpYpZp);
    break;
}
/*
if(AZ<0) { if( (AX==2)|| (AX== -2) ) { p1=r_N[2]; p2=r_N[1]; }
           else { p1=r_N[1]; p2=r_N[2]; }
for(k=0;k<=(r_N[3]-1)/2;k++) { p3=r_N[3]-k;
for(i=0;i<=p1;i++) { for(j=0;j<=p2;j++) {
    p0=F(i,j,k); F(i,j,k)=F[i][j][p3]; F[i][j][p3]=p0; } }
    } // next k
} // end if
switch(AX) {
case -1: p1=r_N[1]/2;
for(k=0;k<=r_N[3];k++) { for(i=0;i<=p1;i++) {
    for(j=0;j<=r_N[2];j++) {
        p0=F(i,j,k); F(i,j,k)=F[r_N[1]-i][r_N[2]-j][k]; F[r_N[1]-i][r_N[2]-j][k]=p0; }
    } // next k
break;
case 2: p1=max(r_N[1],r_N[2]);
for(k=0;k<=r_N[3];k++) {
    for(i=0;i<=(r_N[2]-1)/2;i++) { for(j=0;j<=r_N[1];j++) {
        p0=F(i,j,k); F(i,j,k)=F[r_N[2]-i][j][k]; F[r_N[2]-i][j][k]=p0; } }
    for(i=0;i<=p1;i++) { for(j=0;j<i;j++) {
        p0=F(i,j,k); F(i,j,k)=F[j][i][k]; F[j][i][k]=p0; } }
    } // next k
break;
case -2: p1=max(r_N[1],r_N[2]);
for(k=0;k<=r_N[3];k++) {
    for(i=0;i<=r_N[2];i++) { for(j=0;j<=(r_N[1]-1)/2;j++) {
        p0=F(i,j,k); F(i,j,k)=F[i][r_N[1]-j][k]; F[i][r_N[1]-j][k]=p0; } }
    for(i=0;i<=p1;i++) { for(j=0;j<i;j++) {

```

```

    p0=F(i,j,k); F(i,j,k)=F[j][i][k]; F[j][i][k]=p0; } }
    } // next k
break;
} // end switch
*/
/***** DRAW AND FILL POLYGON *****/
//!! Draw frame ?
/*
if (r_COLc!=0) {setfillstyle(SOLID_FILL,r_COLc) ; fillpoly(6,NX);}
setcolor (r_COLl);drawpoly(7,NX);
line(-un1,r_HI+vn1+vn2,NX[0],NX[1]);
line(-un1,r_HI+vn1+vn2,NX[4],NX[5]);
line(-un1,r_HI+vn1+vn2,NX[8],NX[9]);
*/
clock_start=clock();

/***** PREPARATIONS: *****/
h1=v1/u1 ; h2=v2/u2 ;
IL12=IL[1]; IL13=IL[1]; IL23=IL[2];
IL12*=IL[2]; IL13*=IL[3]; IL23*=IL[3];
r_DIM1=r_DIM-1; r_DIM2=r_DIM*r_DIM ; r_DIM3=r_DIM2*r_DIM ;
for(i=1;i<4;i++) { dt[i]=r_Leye[i]*r_Leye[i]; AP[i]=(double) (0.5-r_N[i]*(double) (IL[i])); }
DL1=-2*IL[1]; DL2=-2*IL[2];
D1=(double) (0.25/IL12); D2=(double) (-0.5*D1/IL[3]);
a=(double) (___min( 30000 , 1000000000./(13.*r_DIM3+43*r_DIM2+10*r_DIM) ));
i=(int) (a/(r_COLb+r_COLf));
DOWN=2*du*LLL ; B2=i*r_COLf*DOWN ; DOWN*=DOWN ;
B1=(double) (i*r_COLb+0.5) ; BD=4*i*(long) (r_DIM3) ;
b=(double) (Na[1]+Na[2]+Na[3]) ; a=(double) (0.001*du*m1/2) ; du*=0.999 ;
CF1=Na[1]*(u1*u1+v1*v1)+Na[2]*v1*v2-b*r_Leye[1]+u1*a; CF1=(Na[1]-CF1)*LL[1]*LLL;
CF2=Na[1]*(u1*u2+v1*v2)+Na[2]*v2*v2-b*r_Leye[2]+u2*a; CF2=(Na[2]-CF2)*LL[2]*LLL;
CF3=v3*(Na[1]*v1+Na[2]*v2)-b*r_Leye[3]; CF3=(Na[3]-CF3)*LL[3]*LLL;
CU1=-LL[1]*u1*du*LLL/r_DIM; CU2=-LL[2]*u2*du*LLL/r_DIM; CV1=-LL[1]*v1*du*LLL/r_DIM;
CV2=-LL[2]*v2*du*LLL/r_DIM; CV3=-LL[3]*v3*du*LLL/r_DIM;
e1=(vn3-vn2)/r_DIM ; e2=(un1*h2-vn1-vn2+vn3)/r_DIM;
e3=vn2/r_DIM ; e4=-un2*h1/r_DIM;
x0=CF1 ; y0=CF2 ; z0=CF3 ;
u1=CU1*r_DIM ; u2=CU2*r_DIM ; v1=CV1*r_DIM ; v2=CV2*r_DIM ; v3=CV3*r_DIM ;
for(i=1;i<4;i++) r_Leye[i]/=r_ALscale[i];
/***** MAIN LOOP: *****/
rfg=0 ;
for (i=0,idim=0;i<=m1;i++,idim+=r_DIM) {
    printf("%3d%", (int) (100*i/m1));
    j0=(int) (___min(e1,e2)+0.99) ; j0=(int) (___max(e3,e4)) ;
    e1=h1 ; e2=h2 ; e3=h2 ; e4=h1 ;
    x[1]=x0+j0*v1 ; x[2]=y0+j0*v2 ; x[3]=z0+j0*v3 ;

    for (j=j0;j<=j1;j++) {
        /* GO INTO CUBE: */
        t=___max( ___max(x[1],x[2]),x[3]) ; ddt=0. ;
        for(k=1;k<4;k++) {
            if(t==x[k]) { NX[k]=r_N[k]; P[k]=-IL[k]; IED[k]=k; TOP[k]=r_N[k]-1; }
            else { a=AP[k]+x[k]-t; if(a<0.5) { td[rig][j]=-1; goto nextj ; }
                AA=(long) a; a-=AA+0.5; NX[k]=AA/(-IL[k]); P[k]=AA+(long) (NX[k])*IL[k];
                if(P[k]==0) { if(NX[k]==0) { ddt+=(a-1)*dt[k]; P[k]=1;
                    IED[k]=0; TOP[k]=0; }
                    else { ddt+=a*dt[k]; P[k]=-IL[k];
                        IED[k]=k; TOP[k]=NX[k]-1; }
                }
            else { IED[k]=0; TOP[k]=NX[k]; ddt+=a*dt[k]; }
        }
        /* next k */
        td[rig][j]=JUMP(t+ddt); /* WE'VE FOUND THE BODY */

        nextj: x[1]+=v1 ; x[2]+=v2 ; x[3]+=v3 ;
    }
    /* next j */

    for(j=0;j<j0;j++) td[rig][j]=-1;
    for(j=j1+1;j<=m2;j++) td[rig][j]=-1;

    /* END OF FINDING ROOTS,LET'S DRAW: */
    if(i==0) {j11=j1; j10=j0; rig=1; mid=0; goto nexti; }
}

```

```

if(i==1) {
    if(jl0<=0)
        wd[0][0]=INTENS(-1,td[0][1],td[1][0],-1,td[0][0]);
    else
        wd[0][jl0]=INTENS(-1,td[0][jl0+1],td[1][jl0],td[0][jl0-1],td[0][jl0]);
        /* putpixel(0,r_HI-jl0*r_DIM,wd[0][jl0]); */
for(j=jl0+1,jdim=r_HI-(jl0+1)*r_DIM;j<=jl1;j++,jdim-=r_DIM) { if(j<m2) t=td[0][j+1];
    else t=-1;
        wd[0][j]=INTENS(-1,t,td[1][j],td[0][j-1],td[0][j]);
        /* putpixel(0,jdim,wd[0][j]); */
    }
for(j=0;j<jl0;j++) { wd[0][j]=0; } for(j=jl1+1;j<=m2;j++) { wd[0][j]=0; }

jm0=j0; jml=j1; lef=0; mid=1; rig=2; p0=0;
goto nexti;
} /* end of case i==1 */

if(i==2) {
    if(jm0<=0)
        wd[1][0]=INTENS(td[lef][0],td[mid][1],td[rig][0],-1,td[mid][0]);
    else
        wd[1][jm0]=INTENS(td[lef][jm0],td[mid][jm0+1],td[rig][jm0],
            td[mid][jm0-1],td[mid][jm0]);
        /* putpixel(r_DIM,r_HI-jm0*r_DIM,wd[1][jm0]); */
for(j=jm0+1,jdim=r_HI-(jm0+1)*r_DIM;j<=jml;j++,jdim-=r_DIM) {
    if(j<m2) t=td[mid][j+1];
    else t=-1;
        wd[1][j]=INTENS(td[lef][j],t,td[rig][j],td[mid][j-1],td[mid][j]);
        /* putpixel(r_DIM,jdim,wd[1][j]); */
    } /* next j */
for(j=0;j<jm0;j++) { wd[1][j]=0; } for(j=jml+1;j<=m2;j++) { wd[1][j]=0; }
jf0=jl0; jf1=jl1; jl0=jm0; jl1=jml; jm0=j0; jml=j1;
m=lef; lef=mid; mid=rig; rig=m;
p1=1; goto nexti;
} /* end of case i==2 */

if(i==3) {
    if(jm0<=0)
        wd[2][0]=INTENS(td[lef][0],td[mid][1],td[rig][0],-1,td[mid][0]);
    else
        wd[2][jm0]=INTENS(td[lef][jm0],td[mid][jm0+1],td[rig][jm0],
            td[mid][jm0-1],td[mid][jm0]);
        /* putpixel(idim-r_DIM,r_HI-jm0*r_DIM,wd[2][jm0]); */
for(j=jm0+1,jdim=r_HI-(jm0+1)*r_DIM;j<=jml;j++,jdim-=r_DIM) {
    if(j<m2) t=td[mid][j+1];
    else t=-1;
        wd[2][j]=INTENS(td[lef][j],t,td[rig][j],td[mid][j-1],td[mid][j]);
        /* putpixel(idim-r_DIM,jdim,wd[2][j]); */
    }
for(j=0;j<jm0;j++) { wd[2][j]=0; } for(j=jml+1;j<=m2;j++) { wd[2][j]=0; }
jmin=__min(jf0,jl0); jmax=__max(jf1,jl1);
for(j=jmin,jdim=jmin*r_DIM; j<=jmax-1; j++,jdim+=r_DIM) {
    if(j==0) {jj5=0;jj6=0;} else {jj5=wd[1][j-1];jj6=wd[0][j-1];}
    if(j==m2-1) {jj1=0;jj2=0;} else {jj1=wd[0][j+2];jj2=wd[1][j+2];}

switch(TEST(wd[0][j],wd[1][j],wd[0][j+1],wd[1][j+1])) {
    case '?':
        CHESS(0,jdim,wd[0][j],wd[1][j],wd[0][j+1],wd[1][j+1],
            jj1,jj2,wd[2][j+1],wd[2][j],jj5,jj6,0,0);
        break;
    case '1':
        GURO2('-',wd[0][j],wd[1][j],wd[0][j+1],wd[1][j+1],
            jj1,jj2,wd[2][j+1],wd[2][j],jj5,jj6,0,0,0,jdim);
        break;
    }
} /* next j */
jf0=jl0; jf1=jl1; jl0=jm0; jl1=jml; jm0=j0; jml=j1;
m=lef; lef=mid; mid=rig; rig=m;
p2=2; p3=3; goto nexti;
} /* end of case i==3 */

else { /* i>3 */
    if(jm0<=0)

```

```

wd[p3][0]=INTENS(td[lef][0],td[mid][1],td[rig][0],-1,td[mid][0]);
else
wd[p3][jm0]=INTENS(td[lef][jm0],td[mid][jm0+1],td[rig][jm0],
td[mid][jm0-1],td[mid][jm0]);
/* putpixel(idim-r_DIM,r_HI-jmin*r_DIM,wd[p3][jm0]); */
for(j=jm0+1,jdim=r_HI-(jm0+1)*r_DIM;j<=jm1;j++,jdim-=r_DIM) {
if(j<m2) t=td[mid][j+1];
else t=-1;
wd[p3][j]=INTENS(td[lef][j],t,td[rig][j],td[mid][j-1],td[mid][j]);
/* putpixel(idim-r_DIM,jdim,wd[p3][j]); */
}
for(j=0;j<jm0;j++) { wd[p3][j]=0; } for(j=jm1+1;j<=m2;j++) { wd[p3][j]=0; }
jmin=__min(jf0,jl0); jmax=__max(jf1,jl1);
for(j=jmin,jdim=jmin*r_DIM; j<=jmax-1; j++,jdim+=r_DIM) {
if(j==0) {jj5=0;jj6=0;} else {jj5=wd[p2][j-1];jj6=wd[p1][j-1];}
if(j==m2-1) {jj1=0;jj2=0;} else {jj1=wd[p1][j+2];jj2=wd[p2][j+2];}

switch(TEST(wd[p1][j],wd[p2][j],wd[p1][j+1],wd[p2][j+1])) {
case '?':
CHESS((i-3)*r_DIM,jdim,wd[p1][j],wd[p2][j],wd[p1][j+1],wd[p2][j+1],
jj1,jj2,wd[p3][j+1],wd[p3][j],jj5,jj6,wd[p0][j],wd[p0][j+1]);
break;
case '1':
GURO2('-',wd[p1][j],wd[p2][j],wd[p1][j+1],wd[p2][j+1],
jj1,jj2,wd[p3][j+1],wd[p3][j],jj5,jj6,wd[p0][j],
wd[p0][j+1],(i-3)*r_DIM,jdim);
break;
}
/* next j */
jf0=jl0; jf1=jl1; jl0=jm0; jl1=jm1; jm0=j0; jm1=j1;
m=lef; lef=mid; mid=rig; rig=m;
m=p0; p0=p1; p1=p2; p2=p3; p3=m;
} /* end of case i>3 */

if(i==m1) {
if(jm0<=0)
wd[p3][0]=INTENS(td[lef][0],td[mid][1],-1,-1,td[mid][0]);
else
wd[p3][jm0]=INTENS(td[lef][jm0],td[mid][jm0+1],-1,td[mid][jm0-1],
td[mid][jm0]);
/* putpixel(idim,r_HI-jmin*r_DIM,wd[p3][jm0]); */
for(j=jm0+1,jdim=r_HI-(jm0+1)*r_DIM;j<=jm1;j++,jdim-=r_DIM) {
if(j<m2) t=td[mid][j+1];
else t=-1;
wd[p3][j]=INTENS(td[lef][j],t,-1,td[mid][j-1],td[mid][j]);
/* putpixel(idim,jdim,wd[p3][j]); */
}
for(j=0;j<jm0;j++) { wd[p3][j]=0; } for(j=jm1+1;j<=m2;j++) { wd[p3][j]=0; }
jmin=__min(jf0,jl0); jmax=__max(jf1,jl1);
for(j=jmin,jdim=jmin*r_DIM; j<=jmax-1; j++,jdim+=r_DIM) {
if(j==0) {jj5=0;jj6=0;} else {jj5=wd[p2][j-1];jj6=wd[p1][j-1];}
if(j==m2-1) {jj1=0;jj2=0;} else {jj1=wd[p1][j+2];jj2=wd[p2][j+2];}

switch(TEST(wd[p1][j],wd[p2][j],wd[p1][j+1],wd[p2][j+1])) {
case '?':
CHESS((m1-2)*r_DIM,jdim,wd[p1][j],wd[p2][j],wd[p1][j+1],wd[p2][j+1],
jj1,jj2,wd[p3][j+1],wd[p3][j],jj5,jj6,wd[p0][j],wd[p0][j+1]);
break;
case '1':
GURO2('-',wd[p1][j],wd[p2][j],wd[p1][j+1],wd[p2][j+1],
jj1,jj2,wd[p3][j+1],wd[p3][j],jj5,jj6,wd[p0][j],
wd[p0][j+1],(m1-2)*r_DIM,jdim);
break;
}
/* next j */

jmin=__min(jm0,j0); jmax=__max(jm1,j1);
for(j=jmin,jdim=jmin*r_DIM; j<=jmax-1; j++,jdim+=r_DIM) {
if(j==0) {jj5=0;jj6=0;} else {jj5=wd[p3][j-1];jj6=wd[p2][j-1];}
if(j==m2-1) {jj1=0;jj2=0;} else {jj1=wd[p2][j+2];jj2=wd[p3][j+2];}

switch(TEST(wd[p2][j],wd[p3][j],wd[p2][j+1],wd[p3][j+1])) {
case '?':

```

```

        CHESS((m1-1)*r_DIM,jdim,wd[p2][j],wd[p3][j],wd[p2][j+1],wd[p3][j+1],
        jj1,jj2,0,0,jj5,jj6,wd[p1][j],wd[p1][j+1]);
        break ;
    case '1':
        GURO2('-',wd[p2][j],wd[p3][j],wd[p2][j+1],wd[p3][j+1],
        jj1,jj2,0,0,jj5,jj6,wd[p1][j],wd[p1][j+1],(m1-1)*r_DIM,jdim);
        break ;
    }
    /* next j */
} /* end of case i==m1 */

nexti: x0+=u1 ; y0+=u2 ;
} /* next i */
if(r_COL1!=0) { ///!! Frame ??
    /*
    setcolor (r_COL1); setlinestyle(SOLID_LINE,0,NORM_WIDTH);
    line (un2,r_HI-vn3,un2,r_HI);
    line (un2,r_HI-vn3,0,r_HI+vn2-vn3);
    line (un2,r_HI-vn3,NX[8],NX[7]);
    */
}

// Finish
for(i=1;i<4;i++) r_Leye[i]*=r_ALscale[i];
clock_end=clock();
return (double)(clock_end-clock_start)/CLOCKS_PER_SEC;
}

/*****
* TEST: test surface presence inside a square
*****/
char RayTracer::TEST(int w1, int w2, int w3, int w4)
{
    if(w1<=0)
    {
        if( (w2<=0) && (w3<=0) && (w4<=0) ) return ('o');
        IS[0][0]=-1;
        return ('?');
    }
    else
    {
        if( (w2>0) && (w3>0) && (w4>0) ) return ('l');
        IS[0][0]=1;
        return ('?');
    }
}

/*****
* Initialize picture parameters
*****/
bool RayTracer::PICTURE(char how)
{
    int i,m1,m2,nx[14];
    double xa[4],a,b,c,u1,u2,v1,v2,v3,du,un1,un2,vn1,vn2,vn3;

    for(i=1;i<4;i++) Na[i]=r_N[i]*r_ALscale[i];
    if(how=='Y')
    {
        for(i=1;i<4;i++) {xa[i]=fabs(r_ALscale[i]/r_Leye[i]);}
        b=__max(xa[2],xa[3]);
        a=__min( FIL/sqrt( __max(xa[1]*b,xa[2]*xa[3]) ) , 15000./__max(xa[1],b) );
        for(i=1;i<4;i++)
        {
            IL[i]=-(int)__max(0.5+a*xa[i],50);
            r_Leye[i]=(100.*r_ALscale[i])/IL[i];
        }
    }
    else
    {

```

```

}

/*****
*
*   Update Ruler pop-up menu
*
*****/
void Ruler::UpdatePopupMenu(CMenu *pop)
{
    GetLength();
    if(r_pix_spacingX<0.0)
    {
        pop->EnableMenuItem(ID_RULER_MM,MF_GRAYED);
        pop->EnableMenuItem(ID_RULER_CM,MF_GRAYED);
        pop->EnableMenuItem(ID_RULER_IN,MF_GRAYED);
        pop->CheckMenuItem (ID_RULER_PIXELS,MF_CHECKED);
    }
    else
    {
        if(r_scale=="mm") pop->CheckMenuItem(ID_RULER_MM,MF_CHECKED );
        else if (r_scale=="cm") pop->CheckMenuItem(ID_RULER_CM,MF_CHECKED );
        else if (r_scale=="in") pop->CheckMenuItem(ID_RULER_IN,MF_CHECKED );
        else pop->CheckMenuItem(ID_RULER_PIXELS,MF_CHECKED );
    }

    CString value;
    value.Format("%.2lf ",r_length);
    pop->ModifyMenu(ID_RULER_VALUE,MF_BYCOMMAND,ID_RULER_VALUE,value+r_scale);
}

/*****
*
*   Compute distance in pixels
*
*****/
void Ruler::GetLength()
{
    r_size=r_end-r_start;
    r_pix_length=hypot(r_size.cx,r_size.cy);
    if(r_scale=="pixels")
        r_length=r_pix_length;
    else
        r_length=r_scale_coeff*hypot(r_size.cx*r_pix_spacingX,
                                     r_size.cy*r_pix_spacingY)/(*r_zoom);
}

/*****
*
*   Increment or decrement the number of tick marks if d!=0
*
*****/
void Ruler::ChangeTicksAndStyle(CDC *pDC, int d)
{
    if(!r_active) return;
    if(d!=0) // change number of tick marks
    {
        int newticks=r_ticks+d;
        if(newticks<0) newticks=0;
        else if(newticks>10) newticks=10;
        if(newticks==r_ticks) return;

        if(r_undo) Draw(pDC);
        r_ticks=newticks;
        Draw(pDC);
    }
}

/*****
*
*   Set distance scale
*
*****/
void Ruler::SetScale(int code)

```

```
{
    if(r_pix_spacingX<0.0) code=4;
    switch(code)
    {
    case 1:
        r_scale="mm";
        r_scale_coeff=1.0;
        break;
    case 2:
        r_scale="cm";
        r_scale_coeff=0.1;
        break;
    case 3:
        r_scale="in";
        r_scale_coeff=1.0/25.4;
        break;
    case 4:
        r_scale="pixels";
        r_scale_coeff=1.0;
        break;
    }
}
```



```

// Selector.h: interface for the Selector class.
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_SELECTOR_H__4E54B653_BCCC_11D2_95F9_00105A21774F__INCLUDED_
#define AFX_SELECTOR_H__4E54B653_BCCC_11D2_95F9_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define SEL_RECTANGLE 0
#define SEL_ELLIPSE 1

class Selector
{
public:
    bool m_undo, m_active;
    int m_shape;

    void Redraw(CDC *pDC, const CPoint& center,
               const CPoint& vertex, const CRect& client);
    void Redraw(CDC *pDC, const CPoint& center);
    void Remove(CDC *pDC);
    void Initialize(CDC* pDC, const CPoint& center, double* zoom,
                  double scaleX, double scaleY, int shape=0);
    CString toString();
    CRect GetRect();
    Selector();
    virtual ~Selector();

private:
    double m_scaleX, m_scaleY, m_area;
    double * m_zoom;
    CRect m_RectSel;

    void Draw(CDC* pDC);
    void Clean();
}

#endif // !defined(AFX_SELECTOR_H__4E54B653_BCCC_11D2_95F9_00105A21774F__INCLUDED_)

```

```
// Selector.cpp: implementation of the Selector class.
```

```
//
```

```
////////////////////////////////////
```

```
#include "stdafx.h"
```

```
#include "DCM.h"
```

```
#include "Selector.h"
```

```
#ifdef _DEBUG
```

```
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
```

```
#define new DEBUG_NEW
```

```
#endif
```

```
////////////////////////////////////
```

```
// Construction/Destruction
```

```
////////////////////////////////////
```

```
Selector::Selector()
```

```
{
```

```
    m_shape=SEL_RECTANGLE;
```

```
    Clean();
```

```
}
```

```
Selector::~Selector()
```

```
{
```

```
    ;
```

```
}
```

```
*****
```

```
Functions to Draw and Update Selector rectangle
```

```
*****
```

```
void Selector::Initialize(CDC *pDC, const CPoint& center, double* zoom,  
    double scaleX, double scaleY, int shape)
```

```
    m_active=true;
```

```
    m_undo=true;
```

```
    m_shape=shape;
```

```
    if (m_shape!=SEL_ELLIPSE)    m_shape=SEL_RECTANGLE;
```

```
    if (scaleX!=0.0)
```

```
    {
```

```
        m_scaleX=m_scaleY=scaleX;
```

```
        if (scaleY!=0.0) m_scaleY=scaleY;
```

```
    }
```

```
    if (zoom)    m_zoom=zoom;
```

```
    m_RectSel=CRect(center.x-128, center.y-128,center.x+127, center.y+127);
```

```
    Draw(pDC);
```

```
}
```

```
void Selector::Draw(CDC *pDC)
```

```
{
```

```
    int    dmode;
```

```
    CPen* old_pen = pDC->SelectObject(&theApp.app_Pen);
```

```
    switch (m_shape)
```

```
    {
```

```
    case SEL_RECTANGLE:
```

```
        dmode=SetROP2(pDC->m_hDC, R2_NOT);
```

```
        pDC->MoveTo(m_RectSel.TopLeft());
```

```
        pDC->LineTo(m_RectSel.right,m_RectSel.top);
```

```
        pDC->LineTo(m_RectSel.right,m_RectSel.bottom);
```

```
        pDC->LineTo(m_RectSel.left,m_RectSel.bottom);
```

```
        pDC->LineTo(m_RectSel.TopLeft());
```

```
        SetROP2(pDC->m_hDC, dmode);
```

```
        break;
```

```
    case SEL_ELLIPSE:
```

```
        dmode=SetROP2(pDC->m_hDC, R2_NOT);
```

```
        pDC->Arc(m_RectSel,m_RectSel.TopLeft(),m_RectSel.TopLeft());
```

```
        SetROP2(pDC->m_hDC, dmode);
```

```
        break;
```

```
    }
```

```
}
```

```

    pDC->SelectObject(old_pen);
}
void Selector::Remove(CDC *pDC)
{
    if(m_undo) Draw(pDC);
    Clean();
}
void Selector::Redraw(CDC *pDC, const CPoint& center,
                     const CPoint& vertex, const CRect& client)    // resize
{
    if(!m_active) return;
    if(m_undo) Draw(pDC); // undo old selection
    CPoint z=vertex-center;
    int a=(abs(z.x)); if(a<16) a=16;
    int b=(abs(z.y)); if(b<16) b=16;
    m_RectSel=CRect(center.x-a, center.y-b,center.x+a, center.y+b);
    m_RectSel.IntersectRect(m_RectSel, client);
    m_RectSel.NormalizeRect();
    if(m_RectSel.IsRectEmpty()) return;
    Draw(pDC); // show new selection
    m_undo=true;
}
void Selector::Redraw(CDC *pDC, const CPoint& center)    // move
{
    if(!m_active) return;
    if(m_undo) Draw(pDC); // undo old selection
    m_RectSel.OffsetRect(-m_RectSel.CenterPoint()+center);
    Draw(pDC); // show new selection
    m_undo=true;
}
/*****
 *
 * Reset selector
 *
 *****/
void Selector::Clean()
{
    m_active=false;
    m_undo=false;
    m_scaleX=m_scaleY=-1.0;
    m_area=0.0;
    double x=1.0; m_zoom = &x;
    m_RectSel=CRect(0,0,1,1);
}
/*****
 *
 * Return Selector region
 *
 *****/
CRect Selector::GetRect()
{
    return m_RectSel;
}
/*****
 *
 * Return Selector info
 *
 *****/
CString Selector::toString()
{
    CString info;
    CString units="pixels";
    double dx=m_RectSel.Width();
    double dy=m_RectSel.Height();
    if(m_scaleX>0 && (*m_zoom)>0)
    {
        units="mm";
        dx = m_scaleX*dx/(*m_zoom);
        dy = m_scaleY*dy/(*m_zoom);
    }
    if(m_shape==SEL_RECTANGLE) m_area=4.0*dx*dy;
}

```

```
else /* ellipse */      m_area=3.1415926*dx*dy;

info.Format("Selected area = %.2lf %s2, %.2lfx%.2lf %s",m_area,units,dx,dy,units);
return info;
}
```

```

#ifndef AFX_SOUNDDIALOG_H__750A3131_C0BF_11D2_9601_00105A21774F__INCLUDED_
#define AFX_SOUNDDIALOG_H__750A3131_C0BF_11D2_9601_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// SoundDialog.h : header file
//

#include <mmsystem.h>

////////////////////////////////////
// SoundDialog dialog

class SoundDialog : public CDialog
{
// Construction
public:
    CString GetWavFileName();
    SoundDialog(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(SoundDialog)
    enum { IDD = IDD_SOUND_DIALOG };
        // NOTE: the ClassWizard will add data members here
    }}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(SoundDialog)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual BOOL OnNotify(WPARAM wParam, LPARAM lParam, LRESULT* pResult);
    }}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(SoundDialog)
    afx_msg void OnSoundPlay();
    virtual BOOL OnInitDialog();
    afx_msg void OnSoundRecord();
    afx_msg void OnSoundStop();
    }}AFX_MSG
    DECLARE_MESSAGE_MAP()

private:
    bool MakeFormatList();
    MMRESULT IsFormatSupported(LPWAVEFORMATEX pwfx, UINT uDeviceID);
    bool SetWaveFormat();
    bool CloseMCI();
    bool SaveToFile();
    bool m_Opened;
    MCI_OPEN_PARMS m_Device;
    CString m_info;
    CString ErrorMCI(DWORD e, CString intro=CString(""));
    bool OpenMCI();
    CString m_FileName;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SOUNDDIALOG_H__750A3137_C0BF_11D2_9601_00105A21774F__INCLUDED_)

```

```

// SoundDialog.cpp : implementation file
//

#include "stdafx.h"
#include "DCM.h"
#include "SoundDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// SoundDialog.cpp : implementation file
//
#include <mmreg.h>
#include <msacm.h>

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// SoundDialog dialog

SoundDialog::SoundDialog(CWnd* pParent /*=NULL*/)
: CDialog(SoundDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(SoundDialog)
    // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}

void SoundDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(SoundDialog)
    // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(SoundDialog, CDialog)
    //{{AFX_MSG_MAP(SoundDialog)
    ON_BN_CLICKED(IDC_SOUND_PLAY, OnSoundPlay)
    ON_BN_CLICKED(IDC_SOUND_RECORD, OnSoundRecord)
    ON_BN_CLICKED(IDC_SOUND_STOP, OnSoundStop)
    ON_MESSAGE(MM_MCINOTIFY, OnNotify)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// SoundDialog message handlers

/*****
 *
 *   Play *.wav file
 *
 *****/
void SoundDialog::OnSoundPlay()
{
    PlaySound(m_FileName, NULL, SND_FILENAME);
}

/*****
 *
 *   Initialize dialog
 *
 *****/
BOOL SoundDialog::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_Opened=false;
    m_FileName=CString("sound.wav");
}

```

```

    return TRUE;
}

/*****
 *
 *   Record sound
 *
 *****/
void SoundDialog::OnSoundRecord()
{
    DWORD dwReturn;

    // Open a waveform-audio device with a new file for recording.
    if(!OpenMCI()) return;

    // Set recording parameters
    //SetWaveFormat();

    MCI_RECORD_PARMS mciRecordParms;
    // Record
    mciRecordParms.dwFrom = 0;
    mciRecordParms.dwTo = 1000;
    mciRecordParms.dwCallback = (DWORD)(this->GetSafeHwnd());
    Beep(900,100);
    if (dwReturn = mciSendCommand(m_Device.wDeviceID, MCI_RECORD,
        MCI_FROM | MCI_NOTIFY, (DWORD)(LPVOID) &mciRecordParms))
    {
        AfxMessageBox(ErrorMCI(dwReturn, "Recording error: "));
        mciSendCommand(m_Device.wDeviceID, MCI_CLOSE, 0, NULL);
        return;
    }
    return;

    *****
    Stop recording/playing
    *****/
void SoundDialog::OnSoundStop()
{
    DWORD dwReturn;
    MCI_GENERIC_PARMS genericParms;
    genericParms.dwCallback = (DWORD)this->GetSafeHwnd();
    if (dwReturn = mciSendCommand(m_Device.wDeviceID, MCI_STOP, MCI_WAIT, (DWORD)(LPVOID) &genericPar
        ms))
    {
        AfxMessageBox(ErrorMCI(dwReturn, "Stop error: "));
        mciSendCommand(m_Device.wDeviceID, MCI_CLOSE, 0, NULL);
    }

    // Save and close MCI
    Beep(500,100);
    SaveToFile();
    CloseMCI();
}

/*****
 *
 *   Catch MM_MCINOTIFY
 *
 *****/
BOOL SoundDialog::OnNotify(WPARAM wParam, LPARAM lParam, LRESULT* pResult)
{
    //AfxMessageBox("On Notified");
    return TRUE;
}

/*****
 *
 *   Open MCI device
 *
 *****/

```

```

*****
bool SoundDialog::OpenMCI()
{
    DWORD dwReturn;

    // Open a waveform-audio device with a new file for recording.
    m_Device.lpstrDeviceType = "waveaudio";
    m_Device.lpstrElementName = "";
    if (dwReturn = mciSendCommand(0, MCI_OPEN, MCI_OPEN_ELEMENT | MCI_OPEN_TYPE,
        (DWORD)(LPVOID) &m_Device))
    {
        AfxMessageBox(ErrorMCI(dwReturn, "Open device error: "));
        m_Opened=false;
        return false;
    }
    m_Opened=true;
    return true;
}

/*****
*
*   Report MCI error
*
*****/
CString SoundDialog::ErrorMCI(DWORD e, CString intro)
{
    CString info;
    char ebuffer[200];
    if(! mciGetErrorString(e, ebuffer, 200) ) info.Format("Unknown error");
    else info=intro+CString(ebuffer);
    info.TrimRight();
    return info;
}

/*****
*
*   Save MCI recording into a file m_FileName
*
*****/
bool SoundDialog::SavetoFile()
{
    DWORD dwReturn;
    MCI_SAVE_PARMS mciSaveParms;
    mciSaveParms.lpfilename = m_FileName;
    if (dwReturn = mciSendCommand(m_Device.wDeviceID, MCI_SAVE,
        MCI_SAVE_FILE | MCI_WAIT, (DWORD)(LPVOID) &mciSaveParms))
    {
        AfxMessageBox(ErrorMCI(dwReturn, "Save file error: "));
        return false;
    }
    return true;
}

/*****
*
*   Close MCI device
*
*****/
bool SoundDialog::CloseMCI()
{
    DWORD dwReturn;
    if (dwReturn = mciSendCommand(m_Device.wDeviceID, MCI_CLOSE, 0, NULL))
    {
        AfxMessageBox(ErrorMCI(dwReturn, "Close error: "));
        mciSendCommand(m_Device.wDeviceID, MCI_CLOSE, 0, NULL);
    }
    return true;
    m_Opened=false;
}

```



```

}

bool SoundDialog::SetWaveFormat()
{
    /*
    DWORD dwReturn;
    MCI_WAVE_SET_PARMS mwspWaveFormParameters;
    mwspWaveFormParameters.wFormatTag=0x0002; //WAVE_FORMAT_DSPGROUP_TRUESPEECH ; //WAVE_FORMAT_AD
PCM ;
    if(dwReturn=mciSendCommand (m_Device.wDeviceID,MCI_SET,
        MCI_WAIT | MCI_WAVE_SET_FORMATTAG ,
        (DWORD) (LPVOID)&mwspWaveFormParameters))
    {
        AfxMessageBox(ErrorMCI(dwReturn, "Set parameters error: "));
        mciSendCommand(m_Device.wDeviceID, MCI_CLOSE, 0, NULL);
    }
    */
    MakeFormatList();

    UINT wReturn;
    PCMWAVEFORMAT pcmWaveFormat;
    // Set up PCMWAVEFORMAT for 11 kHz 8-bit mono.
    pcmWaveFormat.wf.wFormatTag = WAVE_FORMAT_PCM;
    pcmWaveFormat.wf.nChannels = 1;
    pcmWaveFormat.wf.nSamplesPerSec = 11025L;
    pcmWaveFormat.wf.nAvgBytesPerSec = 11025L;
    pcmWaveFormat.wf.nBlockAlign = 1;
    pcmWaveFormat.wBitsPerSample = 8;
    // See if format is supported by any device in system.
    wReturn = IsFormatSupported((WAVEFORMATEX*)&pcmWaveFormat, WAVE_MAPPER);
    // Report results.
    if (wReturn == 0)
        AfxMessageBox("11 kHz 8-bit mono is supported.");
    else if (wReturn == WAVERR_BADFORMAT)
        AfxMessageBox("11 kHz 8-bit mono NOT supported.");
    else
        AfxMessageBox("Error opening waveform device.");

    return true;
}

MMRESULT SoundDialog::IsFormatSupported(LPWAVEFORMATEX pwfx, UINT uDeviceID)
{
    return (waveInOpen(
        NULL,                // ptr can be NULL for query
        uDeviceID,           // the device identifier
        pwfx,                // defines requested format
        NULL,                // no callback
        NULL,                // no instance data
        WAVE_FORMAT_QUERY)); // query only, do not open device
}

bool SoundDialog::MakeFormatList()
{
    CString info;

    int m_iNumDevs=waveInGetNumDevs();
    if(m_iNumDevs==0)
    {
        AfxMessageBox("No input devices found");
        return false;
    }
    WAVEINCAPS* m_pDevCaps=new WAVEINCAPS[m_iNumDevs];
    for(int i=0; i<m_iNumDevs; i++)

```

```

{
    waveInGetDevCaps(i,&m_pDevCaps[i],sizeof(WAVEINCAPS));
    info.Format("wMid=%x,\n wPid=%x,\n szPname=%s\n dwFormats=%ld\n wChannels=%ld",
        m_pDevCaps[i].wMid,m_pDevCaps[i].wPid,
        m_pDevCaps[i].szPname,m_pDevCaps[i].dwFormats,
        m_pDevCaps[i].wChannels);
    AfxMessageBox(info);
}
return true;
}

/*
I used this call to open a MM handle for sample recording

WAVEFORMATEX WINEX;
WAVEIN WIN;

    WINEX.wFormatTag = WAVE_FORMAT_PCM;
    WINEX.nChannels = 2;
    WINEX.nSamplesPerSec = 16000;
    WINEX.wBitsPerSample = 8; // 8,16
    WINEX.nBlockAlign = (WINEX.wBitsPerSample * WINEX.nChannels) / 8;
    WINEX.nAvgBytesPerSec = WINEX.nSamplesPerSec * WINEX.nBlockAlign;
    WINEX.cbSize = 0;

    f = waveInOpen(
        &WIN,
        WAVE_MAPPER,
        &WINEX,
        (unsigned long)waveInProc,
        0,
        CALLBACK_FUNCTION);

I decided to use ADPCM for a better compression , but if I select WAVE_FORMAT_ADPCM to wFormatTag
the waveInOpen returns an error code : 32 . What's happening ?

And of course , if you have a suggestion for a better compression , tell me .

```

Thanks

Accepted Answer

From: chensu Date: Sunday, August 02 1998 - 07:30PM PDT

You need to set the WAVEFORMATEX properly. For example,

```

// Format Tag: WAVE_FORMAT_ADPCM
// Channels: 1
// Samples Per Second: 11,025
// Avg Bytes Per Second: 5,666
// Block Alignment: 256
// Bits Per Sample: 4
// Extra Format Information: 32 bytes
// Offset Data Bytes
0xF4, 0x01, 0x07, 0x00, 0x00, 0x01, 0x00, 0x00,
0x00, 0x02, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00,
0xC0, 0x00, 0x40, 0x00, 0xF0, 0x00, 0x00, 0x00,
0xCC, 0x01, 0x30, 0xFF, 0x88, 0x01, 0x18, 0xFF

```

You may use an utility to convert a PCM wave file to an ADPCM wave file. Then, use the RiffWalk utility (it comes with the Platform SDK) to see its header information.

\*/

```

CString SoundDialog::GetWavFileName()
{

```

```

    CString filename=" ";
    CFile file;

```

```
if( !file.Open(m_FileName, CFile::modeRead) )
{
    AfxMessageBox("Cannot locate sound file");
}
else
{
    filename=file.GetFilePath();
    file.Close();
}
return filename;
}
```

```

#if !defined(AFX_STATISTICS_H__6D7670B1_C129_11D2_8051_000000000000__INCLUDED_)
#define AFX_STATISTICS_H__6D7670B1_C129_11D2_8051_000000000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "Image/Image.h"
// Statistics.h : header file
//

////////////////////
// Statistics dialog

class Statistics : public CDialog
{
// Construction
public:
    int* st_Hist;
    long st_n;
    double st_min, st_max, st_Avg, st_StDev;
    CPoint st_HistMax;

    void Clean();
    bool Analyze(Image* pBmp, CRect* roi=NULL, int shape=0,
        double scaleX=-1.0, double scaleY=-1.0);
    CString toString();
    ~Statistics();
    Statistics(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{AFX_DATA(Statistics)
    enum { IDD = IDD_STATISTICS_DIALOG };
    CString st_numString;
    CString st_sizeString;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(Statistics)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(Statistics)
    virtual BOOL OnInitDialog();
    afx_msg void OnPaint();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()

private:
    int st_shape;
    long st_HistSum;
    double st_area, st_scaleX, st_scaleY;
    CString st_units;
    CRect st_Rect;

    void PaintHistrog(CDC* pDC, CRect r);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STATISTICS_H__6D7670B1_C129_11D2_8051_000000000000__INCLUDED_)

```

```

// Statistics.cpp : implementation file
//

#include "stdafx.h"
#include "DCM.h"
#include "Statistics.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// Statistics dialog

Statistics::Statistics(CWnd* pParent /*=NULL*/)
    : CDialog(Statistics::IDD, pParent)
{
    //{{AFX_DATA_INIT(Statistics)
    st_numString = _T("");
    st_sizeString = _T("");
    st_StDev = 0.0;
    //}}AFX_DATA_INIT
    st_Hist=0;
    Clean();
}

Statistics::~Statistics()
{
    Clean();
}

void Statistics::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(Statistics)
    DDX_Text(pDX, IDC_STAT_AVG, st_Avg);
    DDX_Text(pDX, IDC_STAT_MAX, st_max);
    DDX_Text(pDX, IDC_STAT_MIN, st_min);
    DDX_Text(pDX, IDC_STAT_NUM, st_numString);
    DDX_Text(pDX, IDC_STAT_SIZE, st_sizeString);
    DDX_Text(pDX, IDC_STAT_STD, st_StDev);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(Statistics, CDialog)
    //{{AFX_MSG_MAP(Statistics)
    ON_WM_PAINT()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////
// Statistics message handlers
/*****
*
*   Assign 0 to all statistical parameters
*
*****/
void Statistics::Clean()
{
    st_shape=0; // rectangle by default
    st_Avg=0.0;
    st_max=0.0;
    st_min=0.0;
    st_n=0;
    st_StDev=0.0;
    st_scaleX=st_scaleY=-1.0;
    st_area=0.0;
    st_units="pixels";
    st_Rect=CRect(0,0,0,0);
}

```

```

    st_HistMax=CPoint(0,0);
    st_HistSum=0;
    if(st_Hist)
    {
        delete [] st_Hist;
        st_Hist=0;
    }
}
/*****
*
*   Output string for the status bar
*
*****/
CString Statistics::toString()
{
    CString cs;
    cs.Format("%ld points: [%0.0lf,%0.0lf] range, avg=%0.3lf, dev=%0.3lf",
        st_n, st_min, st_max, st_Avg,st_StDev);
    return cs;
}

/*****
*
*   Perform statistical analysis of the given region in the given image
*
*****/
bool Statistics::Analyze(Image *pBmp,CRect* roi, int shape, double scaleX, double scaleY)
{
    long x, y, dx, dy, p, x0, y0, a, b, ab;
    double temp;

    // Clean all statistical info
    Clean();
    st_shape=shape;
    if(scaleX>0.0 && scaleY>0.0)
    {
        st_scaleX=scaleX;    st_scaleY=scaleY;
        st_units="mm";
    }

    // Set image region
    if(! roi) st_Rect=CRect(0,0,pBmp->GetWidth()-1,pBmp->GetHeight()-1);
    else st_Rect.CopyRect(roi);
    st_Rect.NormalizeRect();

    // Validate
    st_Rect.IntersectRect(st_Rect, CRect(0,0,pBmp->GetWidth()-1,pBmp->GetHeight()-1));
    st_Rect.NormalizeRect();
    if(st_Rect.IsRectEmpty()) return false;

    // Find statistics
    // 1. Average, range and number of points
    st_min=pBmp->GetPixel(st_Rect.left,st_Rect.top);    st_max=st_min;
    st_Avg=0.0; st_n=0;
    if(st_shape==1) // ellipse
    {
        x0=(st_Rect.left+st_Rect.right)/2;  a=st_Rect.right-x0;
        a *= a; if(a==0) a=1;
        y0=(st_Rect.top+st_Rect.bottom)/2;  b=st_Rect.bottom-y0;
        b *= b; if(b==0) b=1;
        ab=a*b;
    }
    for(x=st_Rect.left; x<st_Rect.right; x++)
    {
        for(y=st_Rect.top; y<st_Rect.bottom; y++)
        {
            if(st_shape==1) // ellipse
            {
                if(b*(x-x0)*(x-x0)+a*(y-y0)*(y-y0)>ab) continue;
            }
            p = pBmp->GetLuminance(x,y);
            st_Avg += p;
            if(p>st_max) st_max=p;
            else if (p<st_min) st_min=p;
        }
    }
}

```

```

        st_n ++;
    }
}
if(st_n<=0) return false;
st_Avg /= st_n;
st_area=st_n;
if(st_units=="mm") st_area *= (st_scaleX*st_scaleY);
// 2. Standard deviation and histogram
dx=st_Rect.Width()/80;    if(dx<=0) dx=1;
dy=st_Rect.Height()/80;   if(dy<=0) dy=1;
try { st_Hist=new int[(int)(st_max)+1]; }
catch(...)
{
    AfxMessageBox("Low memory, cannot allocate image histogram",
        MB_OK|MB_ICONEXCLAMATION);
    return false;
}
if(!st_Hist)
{
    AfxMessageBox("Low memory, cannot allocate image histogram",
        MB_OK|MB_ICONEXCLAMATION);
    return false;
}
for(p=0; p<=st_max; p++) st_Hist[p]=0;

st_StDev=0.0; st_HistSum=0;
for(x=st_Rect.left; x<st_Rect.right; x += dx)
{
    for(y=st_Rect.top; y<st_Rect.bottom; y += dy)
    {
        if(st_shape==1) // ellipse
        {
            if(b*(x-x0)*(x-x0)+a*(y-y0)*(y-y0)>ab) continue;
        }
        p=pBmp->GetLuminance(x,y);
        st_Hist[p] ++;
        temp=p-st_Avg;
        st_StDev += temp*temp;
        st_HistSum ++;
    }
}
if(st_HistSum<=0) return false;
st_StDev = sqrt(st_StDev/st_HistSum);
// 3. Find histogram pick (maximum)
st_HistMax=CPoint(0,0);
for(p=0; p<st_max+1; p++)
{
    if(st_Hist[p]>st_HistMax.y)
    {
        st_HistMax.x=p;
        st_HistMax.y=st_Hist[p];
    }
}
return true;
}

/*****
*
* Initialize dialog
*
*****/
BOOL Statistics::OnInitDialog()
{
    if(st_units=="pixels")
    {
        st_sizeString.Format("%d<x<%d, %d<y<%d pixels", st_Rect.left-1,st_Rect.right+1,
            st_Rect.top-1, st_Rect.bottom+1);
        st_numString.Format("%ld",st_n);
    }
    else /* mm */
    {
        double x0=(st_Rect.left-1)*st_scaleX;
        double x1=(st_Rect.right+1)*st_scaleX;
        double y0=(st_Rect.top-1)*st_scaleY;

```

```

double y1=(st_Rect.bottom+1)*st_scaleY;
st_sizeString.Format("%.1lf<x<%.1lf, %.1lf<y<%.1lf mm", x0,x1,y0,y1);
st_numString.Format("%ld, Area: %.1lf mm2",st_n, st_area);
}

CDialog::OnInitDialog();

return TRUE; // return TRUE unless you set the focus to a control
// EXCEPTION: OCX Property Pages should return FALSE
}

/*****
*
* Paint dialog
*
*****/
void Statistics::OnPaint()
{
    CPaintDC dc(this); // device context for painting
    PaintHistog(&dc, CRect(20,180,295,400));

    // Do not call CDialog::OnPaint() for painting messages
}

/*****
*
* Paint histogram in the dialog region r
*
*****/
void Statistics::PaintHistog(CDC *pDC, CRect r)
{
    if(!st_Hist) return;
    long p, pmax;
    pmax=(long)st_max+1;

    // Draw rectangle and axis
    CRect r1=r;
    r1.InflateRect(2,2);
    pDC->Rectangle(r1); // (r1,EDGE_ETCHED,BF_RECT);
    pDC->MoveTo(r.left,r.bottom); pDC->LineTo(r.right, r.bottom);
    pDC->MoveTo(r.left,r.bottom); pDC->LineTo(r.left, r.top);
    pDC->MoveTo(r.left,r.top+2); pDC->LineTo(r.left+3, r.top+2);
    pDC->MoveTo(r.left,r.bottom);

    // Set font
    pDC->SetMapMode(MM_TEXT);
    CFont *oldcf = pDC->SelectObject(&theApp.app_SmallFont);
    pDC->SetTextColor(RGB(0,0,0));
    pDC->SetBkMode(TRANSPARENT);

    // Write titles
    CString title;
    title.Format("%.2lf%%",100*(double)(st_HistMax.y)/st_HistSum);
    pDC->TextOut(r.left+5,r.top,title);
    pDC->TextOut(r.left-1,r.bottom,"0");
    if(pmax-1>st_HistMax.x+10) // max intensity
    {
        title.Format("%d",pmax-1);
        pDC->TextOut(r.right-7,r.bottom,title);
    }
    if(st_HistMax.x>5 || pmax<10) // most frequent intensity
    {
        p=r.left+((long)(st_HistMax.x)*r.Width())/pmax;
        pDC->MoveTo(p,r.bottom); pDC->LineTo(p, r.bottom-10);
        title.Format("%d",st_HistMax.x);
        pDC->TextOut(p-5,r.bottom,title);
    }

    // Plot the histogram
    pDC->MoveTo(r.left,r.bottom);
    for(p=0; p<pmax; p++)
    {
        pDC->LineTo(r.left+(p*r.Width())/pmax,

```



```
        r.bottom-((long)(st_Hist[p])*r.Height())/st_HistMax.y);  
    }  
    // Restore fonts  
    pDC->SelectObject(oldcf);  
}
```

```

#if !defined(AFX_MACPROGRESSCTRL_H_603BBF44_B19C_11D3_90FA_0020AFBC499D__INCLUDED_)
#define AFX_MACPROGRESSCTRL_H_603BBF44_B19C_11D3_90FA_0020AFBC499D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MacProgressCtrl.h : header file
//
// CMacProgressCtrl class, version 1.0
//
// Copyright (c) 1999 Paul M. Meidinger (pmmeidinger@yahoo.com)
//
// Feel free to modify and/or distribute this file, but
// do not remove this header.
//
// I would appreciate a notification of any bugs discovered or
// improvements that could be made.
//
// This file is provided "as is" with no expressed or implied warranty.
//
// History:
//     PMM 12/21/1999      Initial implementation.

////////////////////////////////////
// CMacProgressCtrl window

#include <afxcmn.h>
class CMacProgressCtrl : public CProgressCtrl
{
// Construction
public:
    CMacProgressCtrl();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMacProgressCtrl)
    //}}AFX_VIRTUAL

// Implementation
public:
    BOOL GetIndeterminate();
    void SetIndeterminate(BOOL bIndeterminate = TRUE);
    COLORREF GetColor();
    void SetColor(COLORREF crColor);
    virtual ~CMacProgressCtrl();

    // Generated message map functions
protected:
    //{{AFX_MSG(CMacProgressCtrl)
    afx_msg void OnPaint();
    afx_msg void OnTimer(UINT nIDEvent);
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()
private:
    int m_nIndOffset;
    BOOL m_bIndeterminate;
    void DrawVerticalBar(CDC *pDC, const CRect rect);
    void DrawHorizontalBar(CDC *pDC, const CRect rect);
    void DeletePens();
    void CreatePens();
    CPen m_penColor;
    CPen m_penColorLight;
    CPen m_penColorLighter;
    CPen m_penColorDark;
    CPen m_penColorDarker;
    CPen m_penDkShadow;

```

```

CPen m_penShadow;
CPen m_penLiteShadow;
void GetColors();
COLORREF m_crColor;
COLORREF m_crColorLight;
COLORREF m_crColorLighter;
COLORREF m_crColorLightest;
COLORREF m_crColorDark;
COLORREF m_crColorDarker;
COLORREF m_crDkShadow;
COLORREF m_crShadow;
COLORREF m_crLiteShadow;
};

```

```

////////////////////////////////////

```

```

class OProgress : public CMacProgressCtrl
{
public:
    void ShowProgress(int percent, char* info=NULL);
    bool Initialize();
    OProgress();
    virtual ~OProgress();
};

```

```

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif // !defined(AFX_MACPROGRESSCTRL_H__603BBF44_B19C_11D3_90FA_0020AFBC499D__INCLUDED_)

```

```

// MacProgressCtrl.cpp : implementation file
//
// CMacProgressCtrl class, version 1.0
//
// Copyright (c) 1999 Paul M. Meidinger (pmmeidinger@yahoo.com)
//
// Feel free to modify and/or distribute this file, but
// do not remove this header.
//
// I would appreciate a notification of any bugs discovered or
// improvements that could be made.
//
// This file is provided "as is" with no expressed or implied warranty.
//
// History:
//     PMM 12/21/1999      Initial implementation.

#include "stdafx.h"
#include "MacProgressCtrl.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define IDT_INDETERMINATE      100
#define IND_BAND_WIDTH        20

// Function prototypes.
COLORREF LightenColor(const COLORREF crColor, BYTE byIncreaseVal);
COLORREF DarkenColor(const COLORREF crColor, BYTE byReduceVal);

//-----
//
// COLORREF LightenColor(const COLORREF crColor, BYTE byIncreaseVal)
//
// Return Value:      None.
//
// Parameters       :   crColor - References a COLORREF structure.
//                   byReduceVal - The amount to reduce the RGB values by.
//
// Remarks          :   Lightens a color by increasing the RGB values by the given number.
//
// {
//     BYTE byRed = GetRValue(crColor);
//     BYTE byGreen = GetGValue(crColor);
//     BYTE byBlue = GetBValue(crColor);
//
//     if ((byRed + byIncreaseVal) <= 255)
//         byRed = BYTE(byRed + byIncreaseVal);
//     if ((byGreen + byIncreaseVal) <= 255)
//         byGreen = BYTE(byGreen + byIncreaseVal);
//     if ((byBlue + byIncreaseVal) <= 255)
//         byBlue = BYTE(byBlue + byIncreaseVal);
//
//     return RGB(byRed, byGreen, byBlue);
// } // LightenColorref

//-----
//
// COLORREF DarkenColor(const COLORREF crColor, BYTE byReduceVal)
//
// Return Value:      None.
//
// Parameters       :   crColor - References a COLORREF structure.
//                   byReduceVal - The amount to reduce the RGB values by.
//
// Remarks          :   Darkens a color by reducing the RGB values by the given number.
//
// {
//     BYTE byRed = GetRValue(crColor);
//     BYTE byGreen = GetGValue(crColor);

```

```

    BYTE byBlue = GetBValue(crColor);

    if (byRed >= byReduceVal)
        byRed = BYTE(byRed - byReduceVal);
    if (byGreen >= byReduceVal)
        byGreen = BYTE(byGreen - byReduceVal);
    if (byBlue >= byReduceVal)
        byBlue = BYTE(byBlue - byReduceVal);

    return RGB(byRed, byGreen, byBlue);
} // DarkenColorref

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CMacProgressCtrl

//-----
//
CMacProgressCtrl::CMacProgressCtrl()
//
// Return Value:    None.
//
// Parameters      :    None.
//
// Remarks         :    Standard constructor.
//
{
    m_bIndeterminate = FALSE;
    m_nIndOffset = 0;
    m_crColor = ::GetSysColor(COLOR_HIGHLIGHT);
    GetColors();
    CreatePens();
    // CMacProgressCtrl

//-----
CMacProgressCtrl::~CMacProgressCtrl()
//
// Return Value:    None.
//
// Parameters      :    None.
//
// Remarks         :    None.
//
{
    DeletePens();
    // ~CMacProgressCtrl

BEGIN_MESSAGE_MAP(CMacProgressCtrl, CProgressCtrl)
    //{{AFX_MSG_MAP(CMacProgressCtrl)
    ON_WM_PAINT()
    ON_WM_TIMER()
    ON_WM_ERASEBKGDND()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CMacProgressCtrl message handlers

//-----
//
void CMacProgressCtrl::OnPaint()
//
// Return Value:    None.
//
// Parameters      :    None.
//
// Remarks         :    The framework calls this member function when Windows
//                     or an application makes a request to repaint a portion
//                     of an application's window.
//
{
    CPaintDC dcPaint(this); // device context for painting
    CRect rect, rectClient;

```

```

GetClientRect(rectClient);
rect = rectClient;
BOOL bVertical = GetStyle() & PBS_VERTICAL;

// Create a memory DC for drawing.
CDC dc;
dc.CreateCompatibleDC(&dcPaint);
int nSavedDC = dc.SaveDC();
CBitmap bmp;
bmp.CreateCompatibleBitmap(&dcPaint, rect.Width(), rect.Height());
CBitmap *pOldBmp = dc.SelectObject(&bmp);

CBrush br1(m_crColorLightest);
CBrush br2(::GetSysColor(COLOR_3DFACE));
dc.FillRect(rect, &br2);

int nLower, nUpper;
GetRange(nLower, nUpper);

// Determine the size of the bar and draw it.
if (bVertical)
{
    if (!m_bIndeterminate)
        rect.top = rect.bottom - int(((float)rect.Height() * float(GetPos() - nLower)) / float
(nUpper - nLower));
    dc.FillRect(rect, &br1);
    DrawVerticalBar(&dc, rect);
}
else
{
    if (!m_bIndeterminate)
        rect.right = int(((float)rect.Width() * float(GetPos() - nLower)) / float(nUpper - nLo
wer));
    dc.FillRect(rect, &br1);
    DrawHorizontalBar(&dc, rect);
}

dcPaint.BitBlt(rectClient.left, rectClient.top, rectClient.Width(), rectClient.Height(),
&dc, rectClient.left, rectClient.top, SRCCOPY);

dc.SelectObject(pOldBmp);
dc.RestoreDC(nSavedDC);
dc.DeleteDC();
// OnPaint
}

-----
void CMacProgressCtrl::DrawHorizontalBar(CDC *pDC, const CRect rect)
// Return Value:    None.
// Parameters      :   pDC - Specifies the device context object.
//                   rect - Specifies the rectangle of the progress bar.
// Remarks         :   Draws a horizontal progress bar.
{
    if (!rect.Width())
        return;

    int nLeft = rect.left;
    int nTop = rect.top;
    int nBottom = rect.bottom;

    // Assume we're not drawing the indeterminate state.
    CPen *pOldPen = pDC->SelectObject(&m_penColorLight);

    if (m_bIndeterminate)
    {
        pOldPen = pDC->SelectObject(&m_penColor);
        int nNumBands = (rect.Width() / IND_BAND_WIDTH) + 2;
        int nHeight = rect.Height() + 1;

        int nAdjust = nLeft - IND_BAND_WIDTH + m_nIndOffset;

```

```

{
    // Display pattern rectangle
    Draw(PATTERN_RECT);

    // DoModal
    if(!AllocatePattern()) return IDCANCEL;
    CDialog::DoModal();
    DeallocatePattern();
    Erase(PATTERN_RECT);
    Erase(FOUND_RECT);
    return IDOK;
}

/*****
*
*   Prepare dialog before displaying
*
*****/
BOOL FindRegion::OnInitDialog()
{
    CDialog::OnInitDialog();
    // Force dialog to appear in the left top screen corner
    CRect wr(0,0,f_FoundDisplay.right+10,f_FoundDisplay.bottom+70);
    wr.OffsetRect(theApp.app_ScreenResolution-CSize(50,80)-wr.Size());
    MoveWindow(wr, FALSE);

    // Set parameters
    f_Correlation=-10.0;
    f_ImageStartSearchPoint=CPoint(0,0);
    f_Percent=50;
    f_CDC->SetStretchBltMode(HALFTONE);

    f_Speed.SetRange(0,5,TRUE);
    f_Speed.SetTicFreq(1);
    f_Speed.SetLineSize(2);
    f_Speed.SetPos(3);

    f_Progress.SetRange(0,100);

    return TRUE;
}

/*****
*
*   Display image pattern and found areas on the dialog
*
*****/
void FindRegion::DisplayPatterns()
{
    // Grab dialog CDC
    CDC* dialog_pDC=GetDC();
    dialog_pDC->SetStretchBltMode(HALFTONE);

    // Output and frame image patterns
    f_pBmp->DisplayDIB(dialog_pDC,f_PatternDisplay,f_ImagePatternRect,CPoint(0,0),false);
    if(f_Correlation>=-1.0)
    {
        f_pBmp->DisplayDIB(dialog_pDC,f_FoundDisplay,f_ImageFoundRect,CPoint(0,0),false);
    }
    dialog_pDC->DrawEdge(f_PatternDisplay,EDGE_RAISED,BF_RECT);
    dialog_pDC->DrawEdge(f_FoundDisplay,EDGE_RAISED,BF_RECT);

    // Select font
    dialog_pDC->SetMapMode(MM_TEXT);
    CFont *oldcf = dialog_pDC->SelectObject(&theApp.app_SmallFont);
    dialog_pDC->SetBkColor(RGB(192,192,192));

    // Draw titles
    int off = 12*theApp.app_ResolutionScaleFactor;
    dialog_pDC->TextOut(f_PatternDisplay.left, f_PatternDisplay.top-off,
        "Selected pattern : ");
    if(f_Correlation>=-1.0)
    {
        CString str_corr;

```

```

        str_corr.Format("Match found (%4.0f %%): ",100*f_Correlation);
        dialog_pDC->TextOut(f_FoundDisplay.left, f_FoundDisplay.top-off,str_corr);
    }

    // Reset fonts
    dialog_pDC->SelectObject(oldcf);
}

/*****
*
*   Draw and erase regions
*
*****/
void FindRegion::Draw(UINT code)
{
    CRect tmp_rect;
    switch(code)
    {
        case PATTERN_RECT:
            tmp_rect=f_ScreenPatternRect;
            tmp_rect.DeflateRect(1,1);
            f_CDC->DrawEdge(tmp_rect,EDGE_ETCHED,BF_RECT);
            tmp_rect.DeflateRect(theApp.app_ResolutionScaleFactor,theApp.app_ResolutionScaleFactor);
            f_CDC->DrawEdge(tmp_rect,EDGE_ETCHED,BF_RECT);
            break;
        case FOUND_RECT:
            tmp_rect=f_ScreenFoundRect;
            tmp_rect.DeflateRect(1,1);
            f_CDC->DrawEdge(tmp_rect,EDGE_BUMP,BF_RECT);
            break;
    }
}

void FindRegion::Erase(UINT code)
{
    switch(code)
    {
        case PATTERN_RECT:
            f_pBmp->DisplayDIB(f_CDC,f_ScreenPatternRect,f_ImagePatternRect,
                               CPoint(0,0),false);
            break;
        case FOUND_RECT:
            f_pBmp->DisplayDIB(f_CDC,f_ScreenFoundRect,f_ImageFoundRect,
                               CPoint(0,0),false);
            break;
    }
}

void FindRegion::OnPaint()
{
    CPaintDC dc(this); // device context for painting
    DisplayPatterns();
    // Do not call CDialog::OnPaint() for painting messages
}

/*****
*
*   Allocate and deallocate search pattern
*
*****/
bool FindRegion::AllocatePattern()
{
    int i,j;
    // Allocate search pattern
    int rw=f_ImagePatternRect.Width();
    int rh=f_ImagePatternRect.Height();
    if(rw<=0 || rh<=0) return false;
    try { f_Pattern=new long*[rw]; }
    catch(...) { return false; }
    if(!f_Pattern) return false;
    for(i=0; i<rw; i++)
    {
        f_Pattern[i] = new long[rh];
        if(f_Pattern[i]==0)
        {
            for(j=0; j<i; j++) { if(f_Pattern[j]) delete [] f_Pattern[j]; }
            delete [] f_Pattern;
        }
    }
}

```



```

        return false;
    } // out of memory
}

// Fill search pattern with pixel data
f_Pattern_Average=0;
int x, x0, y, y0, count=0;
for(x0=0,x=f_ImagePatternRect.left; x0<rw; x0++,x++)
{
    for(y0=0,y=f_ImagePatternRect.top; y0<rh; y0++,y++)
    {
        if(f_ReduceNoise) f_Pattern[x0][y0]=f_pBmp->GetSmoothedLuminance(x,y);
        else f_Pattern[x0][y0]=f_pBmp->GetLuminance(x,y);
        f_Pattern_Average += f_Pattern[x0][y0];
        count ++;
    }
}
if(count<1) return false;
f_Pattern_Average /= count;
return true;
}

bool FindRegion::DeallocatePattern()
{
    if(f_Pattern)
    {
        for(int i=0; i<f_ImagePatternRect.Width(); i++)
            if(f_Pattern[i]) delete [] f_Pattern[i];
        delete [] f_Pattern;
        f_Pattern=NULL;
    }
    return true;
}

```

```

#if !defined(AFX_FTPLOGINDIALOG_H__C867C413_D8A3_11D2_8070_000000000000__INCLUDED_)
#define AFX_FTPLOGINDIALOG_H__C867C413_D8A3_11D2_8070_000000000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// FTPLoginDialog.h : header file
//

/////////////////////////////////////////////////////////////////
// FTPLoginDialog dialog
#include "resource.h"
class FTPLoginDialog : public CDialog
{
// Construction
public:
    void SetData(CString host, CString usr, CString pwd);
    FTPLoginDialog(CWnd* pParent = NULL); // standard constructor

// Dialog Data
   //{{AFX_DATA(FTPLoginDialog)
    enum { IDD = IDD_DIALOG_FTP };
    CString m_Host;
    CString m_Pwd;
    CString m_User;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(FTPLoginDialog)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(FTPLoginDialog)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif // !defined(AFX_FTPLOGINDIALOG_H__C867C413_D8A3_11D2_8070_000000000000__INCLUDED_)

```

```

// FTPLoginDialog.cpp : implementation file
//

#include "stdafx.h"
#include "FTPLoginDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// FTPLoginDialog dialog

FTPLoginDialog::FTPLoginDialog(CWnd* pParent /*=NULL*/)
: CDialog(FTPLoginDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(FTPLoginDialog)
    m_Host = _T("");
    m_Pwd = _T("");
    m_User = _T("");
    //}}AFX_DATA_INIT
}

void FTPLoginDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(FTPLoginDialog)
    DDX_Text(pDX, IDC_FTP_HOST, m_Host);
    DDX_Text(pDX, IDC_FTP_PWD, m_Pwd);
    DDX_Text(pDX, IDC_FTP_USER, m_User);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(FTPLoginDialog, CDialog)
    //{{AFX_MSG_MAP(FTPLoginDialog)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// FTPLoginDialog message handlers

void FTPLoginDialog::SetData(CString host, CString usr, CString pwd)
{
    m_Host=host;    m_User=usr;    m_Pwd=pwd;
}

BOOL FTPLoginDialog::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_Pwd="";
    UpdateData(FALSE);
    return TRUE;    // return TRUE unless you set the focus to a control
                   // EXCEPTION: OCX Property Pages should return FALSE
}

```

```

#if !defined(AFX_LOGFILE_H__0B9E5823_7508_11D3_96FD_00105A21774F__INCLUDED_)
#define AFX_LOGFILE_H__0B9E5823_7508_11D3_96FD_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// LogFile.h : header file
//
#include "resource.h"

////////////////////////////////////
// LogFile dialog

class LogFile : public CDialog, public DICOMViewLog
{
// Construction
public:
    afx_msg void    OnClear();
    void            DoModeless(CString win_title="Client Log");
    void            Load(const char *pText) { DICOMViewLog::Load(pText); }
    void            Load(DICOMObject& DO)   { DICOMView::Load(DO); };
    bool            IsOn();
    bool            RefreshText();

    LogFile(CString filename="", RTC* rtc=NULL, CWnd* pParent = NULL)
    {
        CreateDVL((char*)(LPCSTR)filename, rtc);
    };

// Dialog Data
//{{AFX_DATA(LogFile)
enum { IDD = IDD_DIALOG_LOGFILE };
CEdit    m_LogWindow;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(LogFile)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

// Generated message map functions
//{{AFX_MSG(LogFile)
virtual BOOL OnInitDialog();
afx_msg void OnOK();
afx_msg void OnRefresh();
afx_msg void OnClose();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

private:
    void            OnCancel( );
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_LOGFILE_H__0B9E5823_7508_11D3_96FD_00105A21774F__INCLUDED_)

```

```

// LogFile.cpp : implementation file
//

#include "stdafx.h"
#include "DCM.h"
#include "LogFile.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// LogFile dialog
void LogFile::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(LogFile)
    DDX_Control(pDX, IDC_EDIT_FILE_LOG, m_LogWindow);
    DDX_Text(pDX, IDC_EDIT_FILENAME, CString(m_Filename));
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(LogFile, CDialog)
    //{{AFX_MSG_MAP(LogFile)
    ON_BN_CLICKED(IDC_BUTTON_REFRESH, OnRefresh)
    ON_BN_CLICKED(IDC_BUTTON_CLEAR, OnClear)
    ON_BN_CLICKED(IDOK, OnOK)
    ON_WM_CLOSE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// LogFile message handlers

/*****
*
* Set dialog window to log text
*
*****/
bool LogFile::RefreshText()
{
    if(!(this->GetSafeHwnd())) return true;    // dialog is not displayed
    long max_length=10000;
    CString tbuffer((TCHAR)(' '),max_length);

    // Refresh log text
    DICOMViewLog::RefreshText((char*)(LPCSTR)tbuffer, max_length);

    // Display log
    int n=tbuffer.Replace("\n","\r\n");
    m_LogWindow.SetWindowText(tbuffer);
    UpdateData(FALSE);
    m_LogWindow.LineScroll(n+10);
    return true;
}

/*****
*
* Initialize dialog
*
*****/
BOOL LogFile::OnInitDialog()
{
    CDialog::OnInitDialog();
    RefreshText();
    return TRUE;    // return TRUE unless you set the focus to a control
    // EXCEPTION: OCX Property Pages should return FALSE
}

/*****
*
* Display dialog
*****/

```

```

*
*****/
void LogFile::DoModeless(CString win_title /*="Client Log"*/)
{
    if(this->GetSafeHwnd()) return;
    Create(IDD_DIALOG_LOGFILE,NULL);
    // Always display on top
    SetWindowPos(&wndTopMost, 0,0,0,0,
        SWP_NOMOVE | SWP_NOSIZE | SWP_SHOWWINDOW );
    SetWindowText(win_title);
    //ShowWindow(SW_SHOWNORMAL);
    if(win_title.Find("Client")>-1) // offset client window
    {
        theApp.MoveWindowToCorner(this, CSize(20,20));
    }
    else
    {
        theApp.MoveWindowToCorner(this);
    }
}

bool LogFile::IsOn() { return (this->GetSafeHwnd() != NULL); }
*****
*
*   Buttons
*
*****/
void LogFile::OnOK() { DestroyWindow(); }
void LogFile::OnClose() { DestroyWindow(); }
void LogFile::OnCancel() { DestroyWindow(); }
void LogFile::OnRefresh() { RefreshText(); }
void LogFile::OnClear()
{
    DICOMViewLog::RefreshText(NULL, -1);
    if(this->GetSafeHwnd()) m_LogWindow.SetWindowText("");
}

```

```

#if !defined(AFX_LUPA_H__24CE8B94_7D8F_11D2_958F_000000000000__INCLUDED_)
#define AFX_LUPA_H__24CE8B94_7D8F_11D2_958F_000000000000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Lupa.h : header file
//

#include "Image/Image.h"
#include "resource.h"

////////////////////////////////////
// Lupa dialog

class Lupa : public CDialog
{
// Construction
public:
    Lupa(CWnd* pParent = NULL);
    ~Lupa();

// Dialog Data
    //{{AFX_DATA(Lupa)
    enum { IDD = IDD_MAGNIFY_DIALOG };
    private:
        BOOL    m_l_optimize;
        long    m_l_height;
        long    m_l_width;
    public:
        double  m_l_zoom;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(Lupa)
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
public:
    bool    m_l_active;
    CSize   m_l_scnSize;

    void    Initialize(CSize csScn, double scrn_zoom, double lupa_zoom);
    void    Reset_DC(CDC *pDC, CRect& scrolled_client);
    void    Move(CPoint& a, CPoint& rel, Image* pBmp, CDC* pDC);
    bool    Resize(CDC* pDC, CPoint center, CPoint vertex);
    bool    Resize(CDC *pDC);
    CString toString();
    inline CRect Lupa::SetImgRect(CPoint & cpI)
    {
        CRect r( CPoint(cpI.x-(m_l_imgSize.cx>>1), cpI.y-(m_l_imgSize.cy>>1)), m_l_imgSize );
        return r;
    };
    inline CRect SetScrnRect(CPoint & cpS)
    {
        return CRect(CPoint(cpS.x-(this->m_l_scnSize.cx>>1), cpS.y-(this->m_l_scnSize.cy>>1)),
            this->m_l_scnSize);
    };
protected:
    // Generated message map functions
    //{{AFX_MSG(Lupa)
    virtual BOOL OnInitDialog();
    afx_msg void OnChangeMagnifyWidthEdit();
    afx_msg void OnChangeMagnifyHeightEdit();
    afx_msg void OnChangeMagnifyZoomEdit();
    afx_msg void OnChangeMagnifyOptimize();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    bool    m_l_preactive;

```

```
double  l_img_zoom;
CSize   l_imgSize;
CRect   l_scnRect, l_dragRect;
HBITMAP l_DIB;
BYTE*   l_Data;
CDC*    l_DC;
```

```
void     Draw(CDC *pDC);
bool     Update2(CRect& a, CRect&b);
```

```
};
```

```
//{{AFX_INSERT_LOCATION}}
```

```
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.
```

```
#endif // !defined(AFX_LUPA_H__24CE8B94_7D8F_11D2_958F_000000000000__INCLUDED_)
```



```

// Lupa.cpp : implementation file
//

#include "stdafx.h"
#include "DCM.h"
#include "Lupa.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// Lupa dialog
Lupa::Lupa(CWnd* pParent /*=NULL*/)
    : CDialog(Lupa::IDD, pParent)
{
    //{{AFX_DATA_INIT(Lupa)
    l_height = 128*theApp.app_ResolutionScaleFactor;
    l_width = l_height;
    l_zoom = 2.0;
    l_optimize=false;
    //}}AFX_DATA_INIT
    l_active=false;
    l_preactive=false;
    l_DC=NULL;
    l_DIB=NULL;
    l_Data=NULL;
    this->Initialize(CSize(l_width,l_height), 1.0, l_zoom);
}

Lupa::~Lupa()
{
    if(l_DC && !theApp.app_Metheus) delete l_DC;
    if(l_Data) delete [] l_Data;
    if(l_DIB)
    {
        MetheusDeleteCompatibleBitmap(l_DC->GetSafeHdc(),l_DIB);
    }
}

void Lupa::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(Lupa)
    DDX_Text(pDX, IDC_MAGNIFY_HEIGHT_EDIT, l_height);
    DDV_MinMaxLong(pDX, l_height, 0, 1000);
    DDX_Text(pDX, IDC_MAGNIFY_WIDTH_EDIT, l_width);
    DDV_MinMaxLong(pDX, l_width, 0, 1000);
    DDX_Text(pDX, IDC_MAGNIFY_ZOOM_EDIT, l_zoom);
    DDV_MinMaxDouble(pDX, l_zoom, 0., 10.);
    DDX_Check(pDX, IDC_MAGNIFY_OPTIMIZE, l_optimize);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(Lupa, CDialog)
    //{{AFX_MSG_MAP(Lupa)
    ON_EN_CHANGE(IDC_MAGNIFY_WIDTH_EDIT, OnChangeMagnifyWidthEdit)
    ON_EN_CHANGE(IDC_MAGNIFY_HEIGHT_EDIT, OnChangeMagnifyHeightEdit)
    ON_EN_CHANGE(IDC_MAGNIFY_ZOOM_EDIT, OnChangeMagnifyZoomEdit)
    ON_EN_CHANGE(IDC_MAGNIFY_OPTIMIZE, OnChangeMagnifyOptimize)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/*****
* LUPA initialization
*
* Input:
* on-screen lupa size csScn, screen image zoom scrn_zoom, LUPA l_zoom l_img_zoom
*
*****/

```

```

*****/
void Lupa::Initialize(CSize csScn, double scrn_zoom, double lupa_zoom)
{
    l_scnSize=CSize(csScn.cx,csScn.cy);
    l_zoom=lupa_zoom;
    l_img_zoom=scrn_zoom;
    double ivzf=1.0 / (scrn_zoom*lupa_zoom); //combined inversed zoom
    l_imgSize=CSize((long)(1+ivzf*l_scnSize.cx),
                    (long)(1+ivzf*l_scnSize.cy));
    l_dragRect=CRect(0,0,0,0);
}

/*****
*
*   Lupa message handlers
*
*****/
BOOL Lupa::OnInitDialog()
{
    CDialog::OnInitDialog();
    l_width=l_scnSize.cx;
    l_height=l_scnSize.cy;
    UpdateData(FALSE);
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void Lupa::OnChangeMagnifyWidthEdit()
{
    UpdateData(TRUE);
    if(l_width<10) l_width=10;
    l_scnSize.cx=l_width;
    Initialize(l_scnSize, l_img_zoom, l_zoom);
}

void Lupa::OnChangeMagnifyHeightEdit()
{
    UpdateData(TRUE);
    if(l_height<10) l_height=10;
    l_scnSize.cy=l_height;
    Initialize(l_scnSize, l_img_zoom, l_zoom);
}

void Lupa::OnChangeMagnifyZoomEdit()
{
    UpdateData(TRUE);
    if(l_zoom<1.5) l_zoom=1.5;
    Initialize(l_scnSize, l_img_zoom, l_zoom);
}

void Lupa::OnChangeMagnifyOptimize()
{
    UpdateData(TRUE);
    l_optimize=!l_optimize;
}

/*****
*
*   Move lupa over the image CDC, responding to (dis)activated status
*
*****/
void Lupa::Move(CPoint& a, CPoint& rel, Image* pBmp, CDC* pDC)
{
    CPoint p;
    CRect r0,r1;
    CSize da=a-rel;

    if(l_active) // active lupa was requested
    {
        // Remove lupa resizing rectangle, if any
        if(l_dragRect.bottom != 0)
        {
            Draw(pDC);
            l_dragRect=CRect(0,0,0,0);
        }
    }
}

```

```

if(!l_preactive)    // was not active before
{
    r0=CRect(0,0,pBmp->m_ScreenMap.crScreen.Width(),pBmp->m_ScreenMap.crScreen.Height());
    Reset_l_DC(pDC,r0);
    r0=CRect(0,2,0,2); // dummy update area
}
else    r0=l_scnRect;    // Remember previous image area
// Compute new lupa zoom area
r1=SetScrnRect(a);
// For Metheus: ignore rectangles not fully inside the image area
if(theApp.app_Metheus && !pBmp->m_ScreenMap.Screen_in_Image(r1,3))    return;
l_scnRect=r1;
// Compute and redraw update rectangles
bool bu=Update2(r0,r1);
if(theApp.app_Metheus)
{
    r0.InflateRect(2,2);
    pBmp->DisplayDIB(pDC,r0,pBmp->m_ScreenMap.Screen_to_Image(r0),CPoint(da));
    /*
    MetheusLoadImageFromDIB(pDC->GetSafeHdc(),l_DIB,theApp.app_DynamicPaletteStart,
        r0.left,r0.top,r0.Width(),r0.Height(),
        r0.left,r0.top);
    */
}
else
{
    r0.OffsetRect(-da);
    pDC->BitBlt(r0.left,r0.top,r0.Width(),r0.Height(),l_DC,
        r0.left,r0.top,SRCCOPY);
}
if(bu)
{
    if(theApp.app_Metheus)
    {
        r1.InflateRect(2,2);
        pBmp->DisplayDIB(pDC,r1,pBmp->m_ScreenMap.Screen_to_Image(r1),CPoint(da));
        /*
        MetheusLoadImageFromDIB(pDC->GetSafeHdc(),l_DIB,theApp.app_DynamicPaletteStart,
            r1.left,r1.top,r1.Width(),r1.Height(),
            r1.left,r1.top);
        */
    }
    else
    {
        r1.OffsetRect(-da);
        pDC->BitBlt(r1.left,r1.top,r1.Width(),r1.Height(),l_DC,
            r1.left,r1.top,SRCCOPY);
    }
}
// Zoom image into new area
r0=l_scnRect;
r0.OffsetRect(-da);
CRect ir=SetImgRect(pBmp->m_ScreenMap.Screen_to_Image(a));
if(l_optimize)
{
    Image* kadr=new Image(false);    // no pixel undo
    if(!kadr->CreateImage(8*((7+ir.Width())/8),8*((7+ir.Height())/8),
        pBmp->GetBytesPerPixel(), pBmp->m_pPal))
    {
        delete kadr;
        pBmp->DisplayDIB(pDC,r0,ir,CPoint(0,0),false);
        pDC->DrawEdge(&(r0),EDGE_BUMP,BF_RECT);
        return;
    }
    kadr->m_pPal->p_active=false;    // no palettes for lupa !
    pBmp->GetSubimage(kadr,ir.left,ir.top);
    kadr->TR_HistStretch(1, false);
    kadr->DisplayDIB(pDC,r0,CRect(0,0,ir.Width()-1,ir.Height()-1),CPoint(0,0),false);
    delete kadr;
}
else
{
    pBmp->DisplayDIB(pDC,r0,ir,CPoint(0,0),false);
}
}

```

```

    pDC->DrawEdge(&(r0), EDGE_BUMP, BF_RECT);
}
else // disactivated lupa was requested
{
    if(l_preactive)
    {
        p=l_scnRect.TopLeft()-da;
        if(theApp.app_Metheus)
        {
            pBmp->DisplayDIB(pDC, l_scnRect, pBmp->m_ScreenMap.Screen_to_Image(l_scnRect), da);
            /*
            MetheusLoadImageFromDIB(pDC->GetSafeHdc(), l_DIB, theApp.app_DynamicPaletteStart,
                p.x, p.y, l_scnRect.Width(), l_scnRect.Height(),
                p.x, p.y);
            */
        }
        else
        {
            pDC->BitBlt(p.x, p.y, l_scnRect.Width(), l_scnRect.Height(), l_DC,
                p.x, p.y, SRCCOPY);
            delete l_DC;
        }
        l_preactive=false;
        l_DC=0;
    }
}
}

```

```

/*
*****

```

```

Represents update region as two rectangles
Returns false if only one rectangle "a" must be updated
or true if both "a" and "b"
*/

```

```

*****/

```

```

bool Lupa::Update2(CRect & a, CRect & b)
{
    if(a.Width()!=b.Width() || a.Height()!=b.Height() ) return false;
    if(a.EqualRect(&b)) // coinciding rectangles
    {
        a=CRect(a.left, a.top, a.left, a.top);
        return false;
    }
    CRect a1; a1.CopyRect(&a);
    CRect b1; b1.CopyRect(&b);
    if(a.left<=b.left && b.left<=a.right)
    {
        if(a.top<=b.top && b.top<=a.bottom)
        {
            a1=CRect(a.left, a.top, b.left, a.bottom);
            b1=CRect(b.left, a.top, a.right, b.top);
        }
        else if(a.top<=b.bottom && b.bottom<=a.bottom)
        {
            a1=CRect(a.left, a.top, b.left, a.bottom);
            b1=CRect(b.left, b.bottom, a.right, a.bottom);
        }
        else return false;
    }
    else if(a.left<=b.right && b.right<=a.right)
    {
        if(a.top<=b.top && b.top<=a.bottom)
        {
            a1=CRect(a.left, a.top, b.right, b.top);
            b1=CRect(b.right, a.top, a.right, a.bottom);
        }
        else if(a.top<=b.bottom && b.bottom<=a.bottom)
        {
            a1=CRect(a.left, b.bottom, b.right, a.bottom);
            b1=CRect(b.right, a.top, a.right, a.bottom);
        }
        else return false;
    }
}

```

```

    }
    else return false;
    a.CopyRect(&a1);
    b.CopyRect(&b1);
    return true;
}

/*****
*
*   Copies current screen image into lupa CDC l_DC
*
*****/
void Lupa::Reset_l_DC(CDC * pDC, CRect& scrolled_client)
{
    int w=scrolled_client.Width();
    int h=scrolled_client.Height();
    l_preactive=true;
    if(theApp.app_Metheus)
    {
        l_DC=pDC;    return;

        w=2048; h=2560;
        // BYTE buffer
        if(l_Data) { delete [] l_Data; l_Data=NULL; }
        if(l_DIB)
        {
            MetheusDeleteCompatibleBitmap(pDC->GetSafeHdc(),l_DIB);
            l_DIB=NULL;
        }
        l_DIB=MetheusCreateCompatibleBitmap(pDC->GetSafeHdc(),w,h);

        l_Data=new BYTE[w*h*3];
        HDC hdc=pDC->GetSafeHdc();
        BOOL b1=MetheusGetImageIntoData(hdc,l_DIB,theApp.app_DynamicPaletteStart,
                                         (UCHAR*)l_Data, w, h, 2*w, 16,
                                         0,0,w,h,
                                         0,0);

        if(b1==FALSE)
        {
            Beep(700,150);
            AfxMessageBox("MetheusGetImageIntoData Failed");
        }
        BOOL b2=MetheusLoadImageFromData(hdc,l_DIB,theApp.app_DynamicPaletteStart,
                                         l_Data, w, h, 2*w, 16,
                                         0,0,w,h,
                                         scrolled_client.left, scrolled_client.top);

        if(b2==FALSE)
        {
            Beep(700,250);
            AfxMessageBox("MetheusLoadImageFromData Failed");
        }
    }
    else
    {
        if(l_DC) { l_DC->DeleteDC(); delete l_DC; l_DC=0; }
        l_DC=new CDC();
        l_DC->CreateCompatibleDC(pDC);
        CBitmap b;
        b.CreateCompatibleBitmap(pDC,w,h);
        l_DC->SelectObject(&b);
        l_DC->BitBlt(0,0,w,h,pDC,scrolled_client.left, scrolled_client.top, SRCCOPY);
        b.DeleteObject();
    }
}

/*****
*
*   Interactively resize the Lupa region on the image
*
*****/
bool Lupa::Resize(CDC *pDC, CPoint center, CPoint vertex)
{
    Draw(pDC); // remove old rectangle

```

```

    CPoint z=vertex-center;
    int a=(abs(z.x))<<1; if(a<32) a=32;
    int b=(abs(z.y))<<1; if(b<32) b=32;
    Initialize(CSize(a,b), l_img_zoom, l_zoom);
    l_dragRect=SetScrnRect(center);
    Draw(pDC); // draw new rectangle
    return true;
}

bool Lupa::Resize(CDC *pDC)
{
    Draw(pDC); // just remove old rectangle
    l_dragRect=CRect(0,0,0,0);
    return true;
}

/*****
 *
 *   Output Lupa parameters into a string (used in status bar)
 *
 *****/
CString Lupa::toString()
{
    CString s;
    s.Format("Screen size %dx%d, zoom=%.2lf",l_scnSize.cx,l_scnSize.cy,l_zoom);
    return s;
}

/*****
 *
 *   Draw Lupa region rectangle
 *
 *****/
void Lupa::Draw(CDC *pDC)
{
    int dmode=SetROP2(pDC->m_hDC, R2_NOT);
    pDC->MoveTo(l_dragRect.TopLeft());
    pDC->LineTo(l_dragRect.right,l_dragRect.top);
    pDC->LineTo(l_dragRect.right,l_dragRect.bottom);
    pDC->LineTo(l_dragRect.left,l_dragRect.bottom);
    pDC->LineTo(l_dragRect.TopLeft());
    SetROP2(pDC->m_hDC, dmode);
}

```

```

#if !defined(AFX_RENAME_FILE_DIR_DIALOG_H_5527F676_DCE1_11D2_9627_00105A21774F_INCLUDED_)
#define AFX_RENAME_FILE_DIR_DIALOG_H_5527F676_DCE1_11D2_9627_00105A21774F_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Rename_File_Dir_Dialog.h : implementation file
//

#include "stdafx.h"
#include "DCM.h"

//#ifdef _DEBUG
//#define new DEBUG_NEW
//#undef THIS_FILE
//static char THIS_FILE[] = __FILE__;
//endif

////////////////////////////////////
// Rename_File_Dir_Dialog dialog

class Rename_File_Dir_Dialog : public CDialog
{
// Construction
public:
    Rename_File_Dir_Dialog(CString old_name, CWnd* pParent = NULL);    // standard constructor
    CString getName();

// Dialog Data
   //{{AFX_DATA(Rename_File_Dir_Dialog)
    enum { IDD = IDD_DIALOG_RENAME_FILE_DIR };
    CString m_NewName;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(Rename_File_Dir_Dialog)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(Rename_File_Dir_Dialog)
    // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CString m_OldName;
};

////////////////////////////////////
// Rename_File_Dir_Dialog dialog

Rename_File_Dir_Dialog::Rename_File_Dir_Dialog(CString old_name, CWnd* pParent /*=NULL*/)
: CDialog(Rename_File_Dir_Dialog::IDD, pParent)
{
    m_OldName=old_name;
    m_OldName.TrimLeft();
    m_OldName.TrimRight();
    m_OldName.Replace("\\", "/");
    //{{AFX_DATA_INIT(Rename_File_Dir_Dialog)
    m_NewName = m_OldName; //_T("");
    //}}AFX_DATA_INIT
}

/*****
*
*   Take care of the appropriate file/directory name syntax
*
*****/

```

```
*****
```

```
CString Rename_File_Dir_Dialog::getName()
```

```
{
    m_NewName.Remove(' ');
    m_NewName.Remove('/');
    m_NewName.Remove('\\');
    if(m_OldName[0]=='/') m_NewName=CString("/") + m_NewName;
    if(m_OldName.Find(".dcm",-1)>-1 && m_NewName.Find(".dcm",-1)==-1)
        m_NewName += CString(".dcm");
    if(m_OldName.Find(".gz",-1)>-1 && m_NewName.Find(".gz",-1)==-1)
        m_NewName += CString(".gz");
    return m_NewName;
}
```

```
void Rename_File_Dir_Dialog::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{AFX_DATA_MAP(Rename_File_Dir_Dialog)
    DDX_Text(pDX, IDC_EDIT_RENAME, m_NewName);
    DDV_MaxChars(pDX, m_NewName, 200);
    //}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(Rename_File_Dir_Dialog, CDialog)
```

```
//{{AFX_MSG_MAP(Rename_File_Dir_Dialog)
    // NOTE: the ClassWizard will add message map macros here
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Rename_File_Dir_Dialog message handlers
```

```
//{{AFX_INSERT_LOCATION}}
```

```
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
```

```
#endif // !defined(AFX_RENAME_FILE_DIR_DIALOG_H__5527F676_DCE1_11D2_9627_00105A21774F__INCLUDED_)
```



```

// Ruler.h: interface for the Ruler class.
//
////////////////////////////////////

#ifndef AFX_RULER_H_D76C3833_C6DF_11D2_9609_00105A21774F__INCLUDED_
#define AFX_RULER_H_D76C3833_C6DF_11D2_9609_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class Ruler
{
public:
    bool r_undo, r_active;

    void SetScale(int code);
    void ChangeTicksAndStyle(CDC *pDC, int d);
    void UpdatePopupMenu(CMenu* pop);
    void Redraw(CDC* pDC, CPoint& p, double* zoom=NULL, double dx=0.0, double dy=0.0);
    void Draw(CDC* pDC);
    CString toString(CPoint scroll);
    Ruler();
    virtual ~Ruler();

private:
    int r_ticks;
    double r_length, r_pix_length, r_pix_spacingX, r_pix_spacingY, r_scale_coeff;
    double* r_zoom;
    CString r_scale;
    CSize r_size;
    CPoint r_start, r_end;

    void Clean();
    void GetLength();
}

#endif // !defined(AFX_RULER_H_D76C3833_C6DF_11D2_9609_00105A21774F__INCLUDED_)

```

```
// Ruler.cpp: implementation of the Ruler class.
```

```
//
```

```
////////////////////////////////////
```

```
#include "stdafx.h"
```

```
#include "DCM.h"
```

```
#include "Ruler.h"
```

```
#ifdef _DEBUG
```

```
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
```

```
#define new DEBUG_NEW
```

```
#endif
```

```
////////////////////////////////////
```

```
// Construction/Destruction
```

```
////////////////////////////////////
```

```
Ruler::Ruler()
```

```
{
    Clean();
}
```

```
Ruler::~Ruler()
```

```
{
    Clean();
}
```

```
/*
*****
*/
```

```
* Set all to 0
```

```
*****
```

```
void Ruler::Clean()
```

```
{
    r_active=false;
    r_undo=false;
    r_ticks=1;
    r_start=CPoint(30,30);
    r_end=CPoint(10,10);
    r_size=CSize(0,0);
    r_scale=CString("pixels");
    r_pix_spacingX=1.0; r_pix_spacingY=1.0;
    double x=1.0; r_zoom=&x;
    r_scale_coeff=1.0;
    r_length=0.0;
    r_pix_length=0.0;
}
```

```
/*
*****
*/
```

```
* Draw a ruler (once)
```

```
*****
```

```
void Ruler::Draw(CDC *pDC)
```

```
{
    CSize t;
    int dmode=SetROP2(pDC->m_hDC, R2_NOT);
    CPen* old_pen = pDC->SelectObject(&theApp.app_Pen);
    // Draw Ruler line
    pDC->MoveTo(r_start); pDC->LineTo(r_end);
    if(r_ticks)
    {
        // Circle the start point
        pDC->Arc(CRect(r_start.x-6,r_start.y-6,r_start.x+6,r_start.y+6), r_start, r_start);
        // Circle the end point
        if(r_pix_length>16) pDC->Arc(CRect(r_end.x-6,r_end.y-6,r_end.x+6,r_end.y+6), r_end, r_end);

        // Draw tick points
        if(r_pix_length>4*r_ticks)
        {
            t=CSize( (int)(0.5+4*r_size.cy/r_pix_length),

```

```

        -(int)(0.5+4*r_size.cx/r_pix_length) ); // normal vector
CPoint tic;
for (int i=1; i<=r_ticks; i++)
{
    tic=CPoint( (i*r_start.x+(r_ticks-i+1)*r_end.x)/(r_ticks+1),
                (i*r_start.y+(r_ticks-i+1)*r_end.y)/(r_ticks+1));
    pDC->MoveTo(tic+t); pDC->LineTo(tic-t);
}
}
pDC->SelectObject(old_pen);
SetROP2(pDC->m_hDC, dmode);
}

/*****
*
*   Display ruler depending on its status
*
*****/
void Ruler::Redraw(CDC *pDC, CPoint &p, double* zoom, double dx, double dy)
{
    if(!r_active) return;
    // Set pixel spacing scales
    if(dx!=0.0)
    {
        r_pix_spacingY=r_pix_spacingX=dx;
        if(dy!=0.0) r_pix_spacingY=dy;
        if(r_pix_spacingX>0.0 && r_pix_spacingY>0.0)    r_scale="mm";
        else                                            r_scale="pixels";
    }

    // Choose drag point
    double ds=_hypot(r_start.x-p.x,r_start.y-p.y);
    double de=_hypot(r_end.x-p.x,r_end.y-p.y);
    if(ds<de) // start point
    {
        if(ds<5.0) return; // avoid degraded Ruler
        if(r_undo) Draw(pDC); // remove previous ruler
        r_start=CPoint(p.x,p.y);
    }
    else // end point
    {
        if(ds<5.0) return; // avoid degraded Ruler
        if(r_undo) Draw(pDC); // remove previous ruler
        r_end=CPoint(p.x,p.y);
    }

    // Update zoom if needed
    if(zoom)
    {
        r_zoom=zoom;
        int px=(p.x<<1)-10; int py=(p.y<<1)-10;
        if(r_start.x>px) r_start.x=px; if(r_start.y>py) r_start.y=py;
        if(r_end.x>px) r_end.x=px; if(r_end.y>py) r_end.y=py;
    }

    // Draw new ruler
    GetLength();
    Draw(pDC); // new ruler
    r_undo=true;
}

/*****
*
*   Report current measurement for the status bar
*
*****/
CString Ruler::toString(CPoint scroll)
{
    CString info;
    info.Format("Distance from (%d,%d) to (%d,%d) is %.2lf ",
                r_start.x+scroll.x, r_start.y+scroll.y, r_end.x+scroll.x, r_end.y+scroll.y,
                r_length);
    return (info+r_scale);
}

```

```
// FileBrowser.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "FileBrowser.h"
#include "Rename_File_Dir_Dialog.h"
#include "CreateDirectoryDialog.h"
#include "AccurateTimer.h"
#include "Compressor.h"
#include <io.h>
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
// File list types
#define FILE 0
#define DFILE 2
#define DFILE_VU 4
#define DIR 6
#define ROOT 8
```

```
////////////////////////////////////
// FileBrowser dialog
```

```
FileBrowser::FileBrowser(CWnd* pParent /*=NULL*/)
: CDialog(FileBrowser::IDD, pParent), m_InSession("DCM")
```

```
{
    //{{AFX_DATA_INIT(FileBrowser)
    m_Directory = CString("");
    m_NumFiles = 0;
    m_NumFilesTotal = 0;
    m_Directory_FTP = _T("");
    m_NumFiles_FTP = 0;
    m_NumFilesTotal_FTP = 0;
    m_SpeedInfo = _T("");
    //}}AFX_DATA_INIT
    m_ParentDirectory=CString("");
    m_RequestedFile=CString("");
    m_pFTPConnection=NULL;
}
```

```
FileBrowser::~FileBrowser()
```

```
{
    deleteFTP();
    m_InSession.Close();
    m_ImageList.DeleteImageList();
}
```

```
/*
*****
*
* Initialize main parameters
*
*****
*/
```

```
bool FileBrowser::Initialize(CString temp_dir)
{
```

```
    if(m_ImageList.m_hImageList==NULL)
    {
```

```
        m_ImageList.Create(16,17,ILC_COLOR4,9,1);
        /* Initialize Browser icons */
        HICON hicon;
        for(int nic=0; nic<9; nic++)
        {
            hicon=AfxGetApp()->LoadIcon(IDI_FILE);
            m_ImageList.Add(hicon);
        }
```

```
        hicon=AfxGetApp()->LoadIcon(IDI_DFILE);           m_ImageList.Replace(DFILE,hicon);
        hicon=AfxGetApp()->LoadIcon(IDI_DFILE_VU);        m_ImageList.Replace(DFILE_VU,hicon);
        hicon=AfxGetApp()->LoadIcon(IDI_DIR);              m_ImageList.Replace(DIR,hicon);
        hicon=AfxGetApp()->LoadIcon(IDI_ROOT);             m_ImageList.Replace(ROOT,hicon);
        ::DeleteObject(hicon);
```

```

    }

    // Get temp file name
    m_TempFile=temp_dir+CString("/copy_");

    // Initialize hash table of opened files
    m_Opened.InitHashTable(37);

    // Directory parameters
    m_Directory="";
    m_Directory_FTP="";

    // Preset FTP parameters
    m_INhost="";
    m_INlogon="";
    m_INpassword="";

    // Display parameters
    m_DCMonly_loc=true;
    m_DCMonly_FTP=false;
    m_FilterFTP=false;

    return true;
}

void FileBrowser::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(FileBrowser)
    DDX_Control(pDX, IDC_LIST_FTP, m_List_FTP);
    DDX_Control(pDX, IDC_FILE_BROWSE_COMBO, m_DriveList);
    DDX_Control(pDX, IDC_LIST, m_List);
    DDX_Text(pDX, IDC_DIRECTORY, m_Directory);
    DDV_MaxChars(pDX, m_Directory, 200);
    DDX_Text(pDX, IDC_NUMFILES, m_NumFiles);
    DDV_MinMaxInt(pDX, m_NumFiles, -1, 1000000);
    DDX_Text(pDX, IDC_NUMFILES_TOT, m_NumFilesTotal);
    DDV_MinMaxInt(pDX, m_NumFilesTotal, -1, 100000);
    DDX_Text(pDX, IDC_DIRECTORY_FTP, m_Directory_FTP);
    DDV_MaxChars(pDX, m_Directory_FTP, 200);
    DDX_Text(pDX, IDC_NUMFILES_FTP, m_NumFiles_FTP);
    DDV_MinMaxInt(pDX, m_NumFiles_FTP, -1, 10000);
    DDX_Text(pDX, IDC_NUMFILES_TOT_FTP, m_NumFilesTotal_FTP);
    DDV_MinMaxInt(pDX, m_NumFilesTotal_FTP, -1, 10000);
    DDX_Text(pDX, IDC_HOST_FTP, m_INhost);
    DDV_MaxChars(pDX, m_INhost, 100);
    DDX_Text(pDX, IDC_SPEED_FTP, m_SpeedInfo);
    //}}AFX_DATA_MAP
}

```

```

BEGIN_MESSAGE_MAP(FileBrowser, CDialog)
    //{{AFX_MSG_MAP(FileBrowser)
    ON_NOTIFY(NM_DBLCLK, IDC_LIST, OnDbldclkList)
    ON_CBN_SELCHANGE(IDC_FILE_BROWSE_COMBO, OnSelchangeFileBrowseCombo)
    ON_NOTIFY(LVN_COLUMNCLICK, IDC_LIST, OnColumnclickList)
    ON_BN_CLICKED(IDC_BUTTON_FTP_GET, OnButtonFtpGet)
    ON_BN_CLICKED(IDC_BUTTON_FTP_PUT, OnButtonFtpPut)
    ON_COMMAND(ID_LOCALHOST_REFRESH, OnLocRefresh)
    ON_COMMAND(ID_REMOTEHOST_REFRESH, OnFtpRefresh)
    ON_COMMAND(ID_REMOTEHOST_CONNECT, OnGoFtp)
    ON_COMMAND(ID_LOCALHOST_SHOWDICOMONLY, OnLocalhostShowDICOMonly)
    ON_UPDATE_COMMAND_UI(ID_LOCALHOST_SHOWDICOMONLY, OnUpdateLocalhostShowDICOMonly)
    ON_COMMAND(ID_REMOTEHOST_FILTERDICOMFILES, OnRemotehostFilterDICOMfiles)
    ON_UPDATE_COMMAND_UI(ID_REMOTEHOST_FILTERDICOMFILES, OnUpdateRemotehostFilterDICOMfiles)
    ON_COMMAND(ID_REMOTEHOST_SHOWDICOMONLY, OnRemotehostShowDICOMonly)
    ON_UPDATE_COMMAND_UI(ID_REMOTEHOST_SHOWDICOMONLY, OnUpdateRemotehostShowDICOMonly)
    ON_COMMAND(ID_LOCALHOST_DELETE, OnLocalhostDelete)
    ON_COMMAND(ID_REMOTEHOST_DELETE, OnRemotehostDelete)
    ON_COMMAND(ID_LOCALHOST_RENAME, OnLocalhostRename)
    ON_COMMAND(ID_REMOTEHOST_RENAME, OnRemotehostRename)
    ON_NOTIFY(NM_RCLICK, IDC_LIST, OnRclickList)
    ON_COMMAND(ID_LOCALHOST_CREATENEWDIRECTORY, OnLocalhostCreateNewDirectory)
}

```

```

ON_COMMAND(ID_REMOTEHOST_CREATENEWDIRECTORY, OnRemotehostCreateNewDirectory)
ON_COMMAND(ID_REMOTEHOST_OPEN, OnRemotehostOpen)
ON_COMMAND(ID_LOCALHOST_OPEN, OnLocalhostOpen)
ON_UPDATE_COMMAND_UI(ID_LOCALHOST_OPEN, OnUpdateLocalhostOpen)
ON_NOTIFY(NM_RCLICK, IDC_LIST_FTP, OnRclickList)
ON_NOTIFY(NM_DBLCLK, IDC_LIST_FTP, OnDblclkList)
ON_NOTIFY(LVN_COLUMNCLICK, IDC_LIST_FTP, OnColumnclickList)
ON_COMMAND(ID_LOCALHOST_COPYTOREMOTEHOST, OnButtonFtpPut)
ON_COMMAND(ID_REMOTEHOST_COPYTOREMOTEHOST, OnButtonFtpGet)
ON_UPDATE_COMMAND_UI(ID_REMOTEHOST_OPEN, OnUpdateRemotehostOpen)
//}}AFX_MSG_MAP
ON_COMMAND(ID_BROWSER_CLOSE, CDialog::OnCancel)
ON_MESSAGE(WM_KICKIDLE, OnKickIdle)
END_MESSAGE_MAP()

//////////////////////////////////////
// FileBrowser message handlers

/*****
*
*   Global-scope callback function to sort list elements
*
*****/
static int CALLBACK ListCompareFunc(LPARAM lParam1, LPARAM lParam2, LPARAM lParamSort)
{
    CString *l1=(CString*)(DWORD)lParam1;
    CString *l2=(CString*)(DWORD)lParam2;
    if(lParamSort==0)    return l1->CompareNoCase(*l2);
    else return l2->CompareNoCase(*l1);
}

/*****
*
*   Initialize DCM file browser
*
*****/
BOOL FileBrowser::OnInitDialog()
{
    CDialog::OnInitDialog();

    /* Set list icons */
    m_List_FTP.SetImageList(&m_ImageList,LVSIL_SMALL);
    m_List.SetImageList(&m_ImageList,LVSIL_SMALL);

    /* Fill the list of available drives */
    m_DriveList.ResetContent();
    m_DriveList.Dir(DDL_DRIVES | DDL_EXCLUSIVE,"");

    /* Initialize Local Browser columns */
    m_List.InsertColumn(0,NULL,LVCFMT_CENTER,18,0);
    m_List.InsertColumn(1,"Patient Name",LVCFMT_LEFT,100,1);
    m_List.InsertColumn(2,"Study Date",LVCFMT_LEFT,100);
    m_List.InsertColumn(3,"File Name",LVCFMT_LEFT,100);
    m_List.InsertColumn(4,"File Size",LVCFMT_RIGHT,100);
    m_List.InsertColumn(5,"File Creation Date",LVCFMT_RIGHT,100);
    m_List.InsertColumn(6,"Last Access Date",LVCFMT_RIGHT,100);

    /* Initialize FTP Browser columns */
    m_List_FTP.InsertColumn(0,NULL,LVCFMT_CENTER,18,0);
    m_List_FTP.InsertColumn(1,"Patient Name",LVCFMT_LEFT,100,1);
    m_List_FTP.InsertColumn(2,"Study Date",LVCFMT_LEFT,100);
    m_List_FTP.InsertColumn(3,"File Name",LVCFMT_LEFT,100);
    m_List_FTP.InsertColumn(4,"File Size",LVCFMT_RIGHT,100);
    m_List_FTP.InsertColumn(5,"File Creation Date",LVCFMT_RIGHT,100);
    m_List_FTP.InsertColumn(6,"Last Access Date",LVCFMT_RIGHT,100);

    /* Extract current local directory information */
    FindFiles(m_List,false,m_Directory);
    FindFiles(m_List_FTP,true,m_Directory_FTP);
}

```

```

/* enforce row selection in lists */
m_List_FTP.SendMessage( LVM_SETEXTENDEDLISTVIEWSTYLE, 0,LVS_EX_FULLROWSELECT );
m_List.SendMessage( LVM_SETEXTENDEDLISTVIEWSTYLE, 0,LVS_EX_FULLROWSELECT );
// or
//ListView_SetExtendedListViewStyle(m_listControl.m_hWnd, LVS_EX_FULLROWSELECT);

```

```

/* Display updated data */
UpdateData(FALSE);

```

```

return TRUE;
}

```

```

/*****
*
* Extract directory information
*
* Return -1 if fails
* or number of elements found
*
*****/

```

```

int FileBrowser::FindFiles(CListCtrl& myList, bool ftp, CString directory)
{
    BOOL bFound;
    int nitem;
    CString fname, text;
    CTime atime, ctime;
    CFileFind finder;
    CftpFileFind* FTPfinder=NULL;

```

```

// Get FTP connection, if needed

```

```

if(ftp)
{
    if(!resetFTP()) return -1;
    FTPfinder=new CftpFileFind(m_pFTPConnection);
    if(!FTPfinder) return -1;
}

```

```

// Set current directory

```

```

if(ftp)
{
    if(directory=="") m_pFTPConnection->GetCurrentDirectory(directory);
    directory.Replace("\\", "/");
    if(m_pFTPConnection->SetCurrentDirectory(directory)==FALSE)
    {
        AfxMessageBox("Access to "+directory+CString(" denied"),MB_ICONEXCLAMATION | MB_OK);
        delete FTPfinder;
        return -1;
    }
    m_Directory_FTP=CString(directory);
}

```

```

else

```

```

{
    if(directory == "")
    {
        char buffer[200];
        GetCurrentDirectory(200,buffer);
        directory=CString(buffer);
        directory.TrimRight();
    }
    directory.Replace("\\", "/");
    if(SetCurrentDirectory(directory)==FALSE)
    {
        AfxMessageBox("Cannot access directory "+directory,MB_ICONEXCLAMATION | MB_OK);
        return -1;
    }
    m_Directory=CString(directory);
}

```

```

// OK, we can start reading files now - Clean item list

```

```

myList.DeleteAllItems();
int nfilestot=0, num_entries=0;

// Get file info
if(ftp) bFound = FTPfinder->FindFile(CString(directory)+CString("/*"));
else bFound = finder.FindFile(CString(directory)+CString("/*"));
BOOL isdir;
while (bFound)
{
    if(ftp)
    {
        bFound = FTPfinder->FindNextFile();
        isdir=FTPfinder->IsDirectory();
        fname=FTPfinder->GetFileName();
        text.Format("%lu",FTPfinder->GetLength());
        FTPfinder->GetLastAccessTime(ctime);
        FTPfinder->GetCreationTime(ctime);
    }
    else
    {
        bFound = finder.FindNextFile();
        isdir=finder.IsDirectory();
        fname=finder.GetFileName();
        text.Format("%lu",finder.GetLength());
        finder.GetLastAccessTime(ctime);
        finder.GetCreationTime(ctime);
    }
    fname.Replace("\\", "/");
    if(isdir) // directory
    {
        if(fname==".") continue; // do not display current directory
        if(fname=="..")
        {
            nitem=myList.InsertItem(LVIF_PARAM|LVIF_IMAGE,num_entries,NULL,0,0,ROOT,0);
            myList.SetItem(nitem,1,LVIF_TEXT,"<Parent Directory>",0,0,0,0);
            myList.SetItem(nitem,3,LVIF_TEXT,fname+"/",0,0,0,0);
        }
        else
        {
            nitem=myList.InsertItem(LVIF_PARAM|LVIF_IMAGE,num_entries,NULL,0,0,DIR,0);
            myList.SetItem(nitem,1,LVIF_TEXT,"<Subdirectory>",0,0,0,0);
            myList.SetItem(nitem,3,LVIF_TEXT,"/"+fname,0,0,0,0);
        }
        myList.SetItem(nitem,2,LVIF_TEXT,NULL,0,0,0,0);
    }
    else // file
    {
        nfilestot++;
        bool opened;
        // Process compressed or uncompressed DICOM
        if(ftp)
        {
            if(fname[0]=='.') continue; // do not process Unix system files
            opened=OpenedFilesListFind(m_Directory_FTP+CString("/") + fname);
        }
        else opened=OpenedFilesListFind(m_Directory+CString("/") + fname);

        // Grab main file data
        int filetype=FILE;
        if(opened) filetype=DFILE_VU;
        nitem=myList.InsertItem(LVIF_PARAM|LVIF_IMAGE,num_entries,NULL,0,0,filetype,0);
        myList.SetItem(nitem,1,LVIF_TEXT,"",0,0,0,0);
        myList.SetItem(nitem,2,LVIF_TEXT,"",0,0,0,0);
        myList.SetItem(nitem,3,LVIF_TEXT,fname,0,0,0,0);
    }
    myList.SetItemText(nitem,4,text);
    myList.SetItemText(nitem,5,ctime.Format("%d.%m.%Y"));
    myList.SetItemText(nitem,6,atime.Format("%d.%m.%Y"));
    num_entries++;
}

// Get parent directory, clean up
int sl=max(directory.ReverseFind('\\'),directory.ReverseFind('/'));
if(ftp)

```



```

{
    FTPfinder->Close();      delete FTPfinder;
    if(sl<=0)    m_ParentDirectory_FTP=CString(m_Directory_FTP);
    else        m_ParentDirectory_FTP=CString(directory.Left(sl));
    // Set file statistics
    m_NumFilesTotal_FTP=nfilestot;
    // Set parent directory - time and size entries are invalid
    nitem=myList.InsertItem(LVIF_PARAM|LVIF_IMAGE,num_entries,NULL,0,0,ROOT,0);
    myList.SetItem(nitem,1,LVIF_TEXT,"<Parent Directory>",0,0,0,0);
    myList.SetItem(nitem,3,LVIF_TEXT,"../",0,0,0,0);
    myList.SetItem(nitem,2,LVIF_TEXT,NULL,0,0,0,0);
    myList.SetItemText(nitem,4,text); //!!! invalid
    myList.SetItemText(nitem,5,ctime.Format("%d.%m.%Y")); //!!! invalid
    myList.SetItemText(nitem,6,atime.Format("%d.%m.%Y")); //!!! invalid
    resetFTP();
}
else
{
    finder.Close();
    if(sl<=0)    m_ParentDirectory=CString(m_Directory);
    else        m_ParentDirectory=CString(directory.Left(sl));
    // Set file statistics
    m_NumFilesTotal=nfilestot;
    // Set current drive
    int cl=m_Directory.Find(':');
    CString dr=CString(m_Directory.Left(cl));
    dr=CString("[-"+dr+CString("-]");
    if(m_DriveList.SelectString(-1,dr) < 0) return -1;
}

// Locate DICOM files
if(!ftp || (ftp && m_FilterFTP))    DICOM_Filter(myList, ftp);

// Sort
SortByColumn(1,myList);

// Update dialog window and exit
UpdateData(FALSE);
return m_NumFiles;
}

// *****
* Sort browser list "myList" by specified column "col"
*
// *****
void FileBrowser::SortByColumn(int col, CListCtrl& myList)
{
    int entries=myList.GetItemCount();
    if(entries<2 || col<0 || myList.GetColumnWidth(col)<1) return; // Nothing to sort
    CString *keys = new CString[entries];
    if(!keys) return; // Low memory

    int itemState, col_type;
    CString ts, pname;

    // Find column type (in a block, to preserve memory)
    col_type=0;
    {
        char buffer[50];
        LVCOLUMN cl;
        cl.iOrder=col; cl.iSubItem=col; cl.pszText=buffer;
        cl.mask=LVCF_TEXT | LVCF_ORDER | LVCF_SUBITEM;
        myList.GetColumn(col,&cl);
        CString col_name=CString(buffer);
        if(col_name.Find("Date",0)>0) col_type=1; // date type
        else if(col_name.Find("Size",0)>0) col_type=2; // size type
    }

    // Set appropriate sort key
    for(int i=0; i<entries; i++)

```

```

{
    itemState=GetItemImageState(i,myList);
    // Obtain item key string
    ts=CString(myList.GetItemText(i, col));
    if(itemState==FILE || itemState==DFILE || itemState==DFILE_VU) // files
    {
        pname=CString(myList.GetItemText(i, 1));
        if(col_type==1) // date string
            keys[i]=CString(ts.Right(4)+ts.Mid(3,2)+ts.Left(2))+pname;
        else if(col_type==2) // file size
            keys[i]=CString('0',16-ts.GetLength()+ts)+pname;
        else if(col!=1) // name, except patient name
            keys[i]=ts+pname;
        else keys[i]=pname+CString(myList.GetItemText(i, 3));
    }
    else keys[i]=ts; // directories

    // Account for root-directory-file priority adding priority prefix
    if(col_type==1) ts.Format("%d",itemState);
    else ts.Format("%d",9-itemState);
    keys[i] = ts+keys[i];
    myList.SetItemData(i, (DWORD) (&keys[i]));
}
myList.SortItems(ListCompareFunc, (col_type==1));
delete [] keys;
return;
}

/*****
*
* Update state of the item "item" in the list "myList"
*
*****/
int FileBrowser::GetItemImageState(int nitem, CListCtrl& myList)
{
    LVITEM itm;
    itm.iItem=nitem;    itm.mask=LVIF_IMAGE;    // we want to retrieve only image state
    itm.iSubItem=0;
    myList.GetItem(&itm);
    return itm.iImage;
}

int FileBrowser::FindName(CString fname, CListCtrl &myList)
{
    for(int n=0; n<myList.GetItemCount(); n++)
    {
        if(CString(myList.GetItemText(n,3))==fname) return n;
    }
    return -1;
}

void FileBrowser::SetItemImageState(int nitem, int state, CListCtrl& myList)
{
    LVITEM itm;
    itm.iItem=nitem;    itm.mask=LVIF_PARAM|LVIF_IMAGE; // retrieve these parameters
    m_List.GetItem(&itm);
    itm.iImage=state;    itm.iSubItem=0;
    myList.SetItem(&itm);
}

/*****
*
* Get selected position in the file list
*
*****/
int FileBrowser::GetSelectedPosition(bool local)
{
    int sel=-1;
    POSITION pos;
    if(local) pos=m_List.GetFirstSelectedItemPosition();
    else pos=m_List_FTP.GetFirstSelectedItemPosition();
    if (pos == NULL) return -1; // nothing selected

    if(local) sel = m_List.GetNextSelectedItem(pos);
}

```

```

else        sel = m_List_FTP.GetNextSelectedItem(pos);
return sel;
}

/*****
*
*   Process Double-left clicks on the files list
*
*****/
void FileBrowser::OnDblclkList(NMHDR* pNMHDR, LRESULT* pResult)
{
    *pResult = 0;
    bool local= (pNMHDR->idFrom==IDC_LIST);
    int item=GetSelectedPosition(local);
    if(item>=0) OpenFile_or_Directory(local, item);
}

/*****
*
*   Process directory choice list
*
*****/
void FileBrowser::OnSelchangeFileBrowseCombo()
{
    CString drive;
    m_DriveList.GetLBText(m_DriveList.GetCurSel(), drive);
    drive=drive.Mid(2,drive.GetLength()-4);
    CString dcur=CString(m_Directory).Left(drive.GetLength());
    if(drive.CompareNoCase(dcur)==0) return;
    drive.MakeUpper();
    drive=drive+CString(":/");
    FindFiles(m_List, false, drive);
}

/*****
*
*   Sort column when clicked on column header
*
*****/
void FileBrowser::OnColumnclickList(NMHDR* pNMHDR, LRESULT* pResult)
{
    CListCtrl* myList;
    bool local= (pNMHDR->idFrom==IDC_LIST);
    if(local) myList=&m_List;
    else myList=&m_List_FTP;
    NM_LISTVIEW* pnm = (NM_LISTVIEW*)pNMHDR;
    int subitem=pnm->iSubItem;
    if(subitem>0) SortByColumn(subitem, (*myList));
    *pResult = 0;
}

/*****
*
*   Keeping track of opened files
*
*****/
void FileBrowser::OpenedFilesListAdd(CString fullname)
{
    m_Opened[fullname]=1;
}

bool FileBrowser::OpenedFilesListFind(CString fullname)
{
    int val=1;
    return (m_Opened.Lookup(fullname,val)==TRUE);
}

void FileBrowser::OpenedFilesListRemove(CString fullname)
{
    m_Opened.RemoveKey(fullname);
}

```

```

/*****
*
*   Start or close FTP session
*
*****/
void FileBrowser::OnGoFtp()
{
    FTPLoginDialog fld;
    fld.SetData(m_INhost, m_INlogon, m_INpassword);
    if(fld.DoModal() == IDOK)
    {
        if(!resetFTP(fld.m_Host, fld.m_User, fld.m_Pwd)) return;
        FindFiles(m_List_FTP, true);
    }
    return;
}

/*****
*
*   Functions to reset and delete FTP connection
*
*****/
bool FileBrowser::resetFTP(CString host, CString logon, CString pwd)
{
    m_INhost=host; m_INlogon=logon; m_INpassword=pwd;
    if(m_INhost=="") return false;
    try
    {
        deleteFTP();
        m_pFTPConnection=m_InSession.GetFtpConnection(host, logon, pwd);
    }
    catch(CInternetException* pEx)
    {
        TCHAR exCause[255];
        CString info;
        pEx->GetErrorMessage(exCause, 255);
        info=CString(exCause);
        info += _T("Some files may not be displayed.");
        AfxMessageBox(info, MB_ICONEXCLAMATION | MB_OK);
        deleteFTP(); pEx->Delete();
        return false;
    }
    if(!m_pFTPConnection)
    {
        AfxMessageBox("Cannot connect to remote computer", MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    return true;
}

bool FileBrowser::resetFTP()
{
    return resetFTP(m_INhost, m_INlogon, m_INpassword);
}

void FileBrowser::deleteFTP()
{
    if(m_pFTPConnection)
    {
        m_pFTPConnection->Close();
        delete m_pFTPConnection;
        m_pFTPConnection=NULL;
    }
}

/*****
*
*   Locate DICOM images in the given list
*   based on file contents
*
*****/

```

```

void FileBrowser::DICOM_Filter(CListCtrl &myList, bool ftp)
{
    int         nstate, ndcm_files=0;
    char         ctmp[64];
    CString      fname, locname, fnamepath,tmp;
    DICOMDataObject dob;

    // Filter unclassified files only
    for(int n=0; n<myList.GetItemCount(); n++)
    {
        nstate=GetItemImageState(n,myList);
        if(nstate!=FILE && nstate!=DFILE_VU) continue;
        fname=CString(myList.GetItemText(n, 3));

        // Create local file, if ftp connection
        if(ftp)
        {
            fnamepath=m_Directory_FTP+CString("/") + fname;
            locname=m_TempFile+fname;
            if(!FileTransfer(fnamepath,locname,true,
                DICOMObject::HighestPreviewFileSize)) continue;
        }
        else    locname=fnamepath=m_Directory+CString("/") + fname;

        // Uncompress if needed
        if(locname.Find(".gz",0)>0) // compressed DICOM
        {
            tmp=m_TempFile+fname+CString("_unzip");
            if(!Compressor::Z_unCompress(locname,tmp,1)) continue;
            locname=tmp;
        }

        // Load DICOM object in preview mode
        dob.Reset();
        if(!dob.LoadFromFile((char*)(LPCSTR)locname,true))
        {
            if( (!ftp && m_DCMOnly_loc) || (ftp && m_DCMOnly_FTP))
            {
                myList.DeleteItem(n);    n--;
            }
            continue;
        }
        DICOMRecord dfr;
        dfr.SetRecord(dob,"");
        myList.SetItem(n,1,LVIF_TEXT,dfr.GetPatientName(),0,0,0,0);
        dfr.FormatStudyDate(ctmp,64,false);
        myList.SetItem(n,2,LVIF_TEXT,ctmp,0,0,0,0);
        if(nstate!=DFILE_VU) SetItemImageState(n,DFILE,myList);
        ndcm_files++;
        if(n%5==0)
        {
            myList.RedrawItems(1,n);    myList.UpdateWindow();
        }
    }
    if(ftp) m_NumFiles_FTP=ndcm_files;
    else    m_NumFiles=ndcm_files;
}

/*****
*
*   Refresh current FTP Window
*
*****/
void FileBrowser::OnFtpRefresh()
{
    if(!resetFTP()) return;
    FindFiles(m_List_FTP, true, m_Directory_FTP);
}

void FileBrowser::OnLocRefresh()
{
    FindFiles(m_List, false, m_Directory);
}

```

```

}

/*****
*
*   Get a binary file from remote server
*
*****/
void FileBrowser::OnButtonFtpGet()
{
    POSITION pos = m_List_FTP.GetFirstSelectedItemPosition();
    if (pos == NULL)
    {
        AfxMessageBox("Please select a remote file first\nby clicking on its icon",MB_ICONEXCLAMATION | MB_OK);
        return;
    }
    int n = m_List_FTP.GetNextSelectedItem(pos);
    int nstatus=GetItemImageState(n,m_List_FTP);
    if(nstatus != FILE && nstatus != DFILE && nstatus != DFILE_VU)
    {
        AfxMessageBox("Cannot copy directories",MB_ICONEXCLAMATION | MB_OK);
        return;
    }
    CString ftp_fname = CString(m_List_FTP.GetItemText(n,3));
    CString loc_fname=m_Directory+"/"+ftp_fname;
    if(FindName(ftp_fname,m_List)>0)
    {
        if(AfxMessageBox("Overwrite "+loc_fname+" ?",
                        MB_ICONEXCLAMATION | MB_YESNO)==IDNO) return;
    }
    FileTransfer(m_Directory_FTP+"/"+ftp_fname,loc_fname,true,-1);
    OnLocRefresh();
    Beep(500,50);
}

/*****
*
*   Put a binary file to a remote server
*
*****/
void FileBrowser::OnButtonFtpPut()
{
    POSITION pos = m_List.GetFirstSelectedItemPosition();
    if (pos == NULL)
    {
        AfxMessageBox("Please select a local file first\nby clicking on its icon",MB_ICONEXCLAMATION | MB_OK);
        return;
    }
    int n = m_List.GetNextSelectedItem(pos);
    int nstatus=GetItemImageState(n,m_List);
    if(nstatus != FILE && nstatus != DFILE && nstatus != DFILE_VU)
    {
        AfxMessageBox("Cannot copy directories",MB_ICONEXCLAMATION | MB_OK);
        return;
    }
    CString loc_fname = CString(m_List.GetItemText(n,3));
    CString ftp_fname=m_Directory_FTP+"/"+loc_fname;
    if(FindName(loc_fname,m_List_FTP)>0)
    {
        if(AfxMessageBox("Overwrite "+ftp_fname+" ?",
                        MB_ICONEXCLAMATION | MB_YESNO)==IDNO) return;
    }
    if(!FileTransfer(ftp_fname,m_Directory+"/"+loc_fname,false,-1))
    {
        AfxMessageBox("Cannot copy. \nMake sure you have remote host connected",MB_ICONEXCLAMATION | MB_OK);
        return;
    }
    OnFtpRefresh();
    Beep(400,50);
}

```

```

/*****
*
*   Handle all kinds of file transfers
*   between local and remote computers
*
*****/
bool FileBrowser::FileTransfer(CString ftp_name, CString loc_name, bool to_local,
                               int size)
{
    static bool failed=false;
    BOOL done;
    double vremia;
    if(size==0) return true;    // empty file
    CInternetFile* ifile=NULL;
    _iobuf* floc;
    BYTE* bf=NULL;

    // Refresh FTP connection
    if(!resetFTP()) return false;

    // Try to copy
    try
    {
        if(size<0) // Transfer the entire file at once
        {
            long fsize=0;
            AccurateTimer ac;  ac.Begin();
            if(to_local) done=m_pFTPConnection->GetFile(ftp_name,loc_name,FALSE);
            else         done=m_pFTPConnection->PutFile(loc_name, ftp_name);
            // Find size of the copied file
            {
                _iobuf* t_file;
                t_file=fopen(loc_name,"rb");
                if (t_file != NULL)
                {
                    fsize = _filelength(_fileno(t_file)); // local file size
                    fclose(t_file);
                }
            }
            vremia=ac.End();
            if(vremia>0.1 && fsize>0)
            {
                m_SpeedInfo.Format(" %.3lf  Kbytes/sec",fsize/(1000.0*vremia));
                UpdateData(FALSE);
            }
            return (done==TRUE);
        }
        else if(size>0)
        {
            try { bf=new BYTE[size]; }
            catch(...) { return false; }
            if(!bf) return false;
            // Open local and remote files
            if(to_local)
            {
                ifile=m_pFTPConnection->OpenFile(ftp_name,GENERIC_READ,
                                                  FTP_TRANSFER_TYPE_BINARY,1);
                floc=fopen(loc_name,"wb");
            }
            else
            {
                ifile=m_pFTPConnection->OpenFile(ftp_name,GENERIC_WRITE,
                                                  FTP_TRANSFER_TYPE_BINARY,1);
                floc=fopen(loc_name,"rb");
            }
            bool can_transfer = (floc!=NULL && ifile!=NULL);
            if(can_transfer)
            {
                int lu;
                if(to_local)
                {
                    lu=ifile->Read(bf,size);
                    fwrite(bf,1,lu,floc);
                }
            }
        }
    }
}

```

```

        else
        {
            lu=fread(bf,1,size,floc);
            ifile->Write(bf,lu);
        }
        delete [] bf;    bf=NULL;
        fclose(floc);    ifile->Close(); delete ifile;
        return can_transfer;
    }
}
catch(CInternetException* ex)
{
    if(!failed)
    {
        ex->ReportError(MB_ICONEXCLAMATION | MB_OK);
        failed=true;
    }
    ex->Delete();
    if(bf) delete [] bf;
    if(ifile)
    {
        try { ifile->Close(); } catch (CInternetException* ex2) { ex2->Delete(); }
        delete ifile;
    }
    return false;
}
return false;
}

/*****
*
* Local: "Show DICOMs only" message handler
*
*****/
void FileBrowser::OnLocalhostShowDICOMonly()
{
    m_DCOMonly_loc = ! m_DCOMonly_loc;
    OnLocRefresh();
    Beep(500,50);
}

void FileBrowser::OnUpdateLocalhostShowDICOMonly(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_DCOMonly_loc);
}

/*****
*
* Remote: "Filter DICOM" message handler
*
*****/
void FileBrowser::OnRemotehostFilterDICOMfiles()
{
    m_FilterFTP = !m_FilterFTP;
    OnFtpRefresh();
    Beep(400,50);
}

void FileBrowser::OnUpdateRemotehostFilterDICOMfiles(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_FilterFTP);
}

/*****
*
* Remote: "Show DICOM only" message handler
*
*****/
void FileBrowser::OnRemotehostShowDICOMonly()
{
    if(!m_FilterFTP) return;
    m_DCOMonly_FTP = ! m_DCOMonly_FTP;
    OnFtpRefresh();
    Beep(400,50);
}

```



```

}
void FileBrowser::OnUpdateRemoteHostShowDICOMonly(CCmndUI* pCmdUI)
{
    pCmdUI->Enable(m_FilterFTP);
    pCmdUI->SetCheck(m_DCMonly_FTP && m_FilterFTP);
}

```

```

/*****
*

```

```

*   Rename selected file/directory on the current computer
*   (local or FTP)
*

```

```

*****/
bool FileBrowser::RenameFile_or_Directory(CListCtrl &myList, bool is_local)
{

```

```

    POSITION pos = myList.GetFirstSelectedItemPosition();
    if (pos == NULL)
    {
        AfxMessageBox("No item selected",MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    int n = myList.GetNextSelectedItem(pos);
    int nstatus=GetItemImageState(n,myList);
    bool isfile=(nstatus == FILE || nstatus == DFILE || nstatus == DFILE_VU);
    if(!is_local && !isfile)
    {
        AfxMessageBox("Cannot rename remote directories",MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    BOOL done;
    CString name =CString(myList.GetItemText(n,3));
    Rename_File_Dir_Dialog rfdd(name);
    if(rfdd.DoModal()==IDCANCEL) return false;
    if(is_local)
    {
        if(isfile) done=MoveFile(m_Directory+"/"+name,m_Directory+"/"+rfdd.getName());
        else done=MoveFile(m_Directory+name,m_Directory+rfdd.getName());
    }
    else
    {
        if(!resetFTP()) return false;
        if(isfile) done=m_pFTPConnection->Rename(m_Directory_FTP+"/"+name,
            m_Directory_FTP+"/"+rfdd.m_NewName);
    }
    if(done==FALSE)
    {
        AfxMessageBox("Cannot rename "+name,MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    else
    {
        if(is_local) OnLocRefresh();
        else OnFtpRefresh();
    }
    return true;
}

```

```

/*****
*

```

```

*   Create a new directory (local or ftp)
*

```

```

*****/
bool FileBrowser::CreateNewDirectory(CListCtrl &myList, bool is_local)
{

```

```

    CreateDirectoryDialog cdd;

```

```

if(cdd.DoModal()==IDCANCEL) return false;
CString name=cdd.getName();
if(name.IsEmpty())
{
    AfxMessageBox("Cannot create: empty name",MB_ICONEXCLAMATION | MB_OK);
    return false;
}
if(FindName(name, myList)>-1)
{
    AfxMessageBox("Cannot create: already exists",MB_ICONEXCLAMATION | MB_OK);
    return false;
}
BOOL done;
if(is_local)    done=CreateDirectory(m_Directory+name, NULL);
else
{
    if(!resetFTP()) return false;
    done=m_pFTPConnection->CreateDirectory(m_Directory_FTP+name);
}
if(done==FALSE)
{
    AfxMessageBox("Cannot create "+name,MB_ICONEXCLAMATION | MB_OK);
    return false;
}
else
{
    if(is_local)    OnLocRefresh();
    else            OnFtpRefresh();
}
return true;
}

void FileBrowser::OnLocalhostCreateNewDirectory()
{
    CreateNewDirectory(m_List, true);
    Beep(500,50);
}

void FileBrowser::OnRemotehostCreateNewDirectory()
{
    CreateNewDirectory(m_List_FTP, false);
    Beep(400,50);
}

/*****
*
*   Open selected file/directory from the current computer
*   (local or FTP)
*
*****/
void FileBrowser::OnLocalhostOpen()
{
    POSITION pos = m_List.GetFirstSelectedItemPosition();
    if (pos == NULL)    return; // nothing selected

    int item = m_List.GetNextSelectedItem(pos); //single selection
    OpenFile_or_Directory(true, item);
}

void FileBrowser::OnUpdateLocalhostOpen(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_List.GetFirstSelectedItemPosition()!=NULL);
}

```

```

void FileBrowser::OnRemoteHostOpen()
{
    POSITION pos = m_List_FTP.GetFirstSelectedItemPosition();
    if (pos == NULL) return; // nothing selected

    int item = m_List_FTP.GetNextSelectedItem(pos); //single selection
    OpenFile_or_Directory(false, item);
}

void FileBrowser::OnUpdateRemoteHostOpen(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_List_FTP.GetFirstSelectedItemPosition() != NULL);
}

void FileBrowser::OpenFile_or_Directory(bool local, int item)
{
    if(item < 0) return; // something is wrong
    CListCtrl* myList;
    if(local) myList = &m_List;
    else myList = &m_List_FTP;

    // Update file browser
    m_RequestedFile = CString("/") + CString(myList->GetItemText(item, 3));
    switch(GetItemImageState(item, (*myList)))
    {
        case FILE:
        case DFILE_VU:
        case DFILE:
            SetItemImageState(item, DFILE_VU, (*myList));
            if(local)
            {
                m_RequestedFile = CString(m_Directory) + m_RequestedFile;
                OpenedFilesListAdd(m_RequestedFile);
            }
            else
            {
                CString tm = CString(m_Directory_FTP) + m_RequestedFile;
                OpenedFilesListAdd(tm);
                m_RequestedFile = m_TempFile + CString(myList->GetItemText(item, 3));
                if(!FileTransfer(tm, m_RequestedFile, true, -1)) return; // cannot download
            }
            CDialog::OnOK();
            break; // unchecked file
        case DIR:
            if(local) FindFiles((*myList), false, CString(CString(m_Directory) + CString(myList->GetItemText(item, 3))));
            else FindFiles((*myList), true, CString(CString(m_Directory_FTP) + CString(myList->GetItemText(item, 3))));
            return; // switch to subdirectory
        case ROOT:
            if(local) FindFiles((*myList), false, m_ParentDirectory);
            else FindFiles((*myList), true, m_ParentDirectory_FTP);
            return; // switch to parent directory
        default: return; // do nothing
    }
    myList->Update(item);
}

/*****
*
* Delete selected file/directory from the current computer
* (local or FTP)
*
*****/
bool FileBrowser::DeleteFile_or_Directory(CListCtrl &myList, bool is_local)
{
    POSITION pos = myList.GetFirstSelectedItemPosition();
    if (pos == NULL)
    {
        AfxMessageBox("No item selected", MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    int n = myList.GetNextSelectedItem(pos);
    int nstatus = GetItemImageState(n, myList);
    bool isfile = (nstatus == FILE || nstatus == DFILE || nstatus == DFILE_VU);
    BOOL done;
}

```

```

CString name =CString(myList.GetItemText(n,3));
if( AfxMessageBox("Do you really want to delete "+name+" ?",
    MB_ICONEXCLAMATION | MB_YESNO) == IDNO ) return false;
if(is_local)
{
    if(isfile)        done=DeleteFile(m_Directory+"/"+name);
    else              done=RemoveDirectory(m_Directory+name);
}
else
{
    if(!resetFTP()) return false;
    if(isfile)        done=m_pFTPConnection->Remove(m_Directory_FTP+"/"+name);
    else              done=m_pFTPConnection->RemoveDirectory(m_Directory_FTP+name);
}
if(done==FALSE)
{
    AfxMessageBox("Cannot delete "+name,MB_ICONEXCLAMATION | MB_OK);
    return false;
}
else
{
    if(is_local)      OnLocRefresh();
    else              OnFtpRefresh();
}
return true;
}

void FileBrowser::OnLocalhostRename()
{
    RenameFile_or_Directory(m_List, true);
    Beep(500,50);
}

void FileBrowser::OnRemotehostRename()
{
    RenameFile_or_Directory(m_List_FTP, false);
    Beep(400,50);
}

/*****
*
* Local: "Delete" message handler
*
*****/
void FileBrowser::OnLocalhostDelete()
{
    DeleteFile_or_Directory(m_List, true);
    Beep(500,50);
}

/*****
*
* Remote: "Delete" message handler
*
*****/
void FileBrowser::OnRemotehostDelete()
{
    DeleteFile_or_Directory(m_List_FTP, false);
    Beep(400,50);
}

/*****
*
* Support for OnUpdate messages in the menu
*
*****/
LRESULT FileBrowser::OnKickIdle(WPARAM, LPARAM)
{
    CMenu* pMainMenu = GetMenu();
    if(!pMainMenu) return FALSE;
    CCmdUI cmdUI;

```

```

for (UINT n = 0; n < pMainMenu->GetMenuItemCount(); ++n)
{
    CMenu* pSubMenu = pMainMenu->GetSubMenu(n);
    if(!pSubMenu) continue;
    cmdUI.m_nIndexMax = pSubMenu->GetMenuItemCount();
    for (UINT i = 0; i < cmdUI.m_nIndexMax; ++i)
    {
        cmdUI.m_nIndex = i;
        cmdUI.m_nID = pSubMenu->GetMenuItemID(i);
        cmdUI.m_pMenu = pSubMenu;
        cmdUI.DoUpdate(this, FALSE);
    }
}
return TRUE;
}

```

```

/*****
*
*   Dynamic popup, context-sensitive menu
*   for browser list items
*
*****/
void FileBrowser::OnRclickList(NMHDR* pNMHDR, LRESULT* pResult)
{
    // Grab necessary parameters
    CListCtrl* myList;
    bool local= (pNMHDR->idFrom==IDC_LIST);
    if(local) myList=&m_List; else myList=&m_List_FTP;
    // Click positioning
    CPoint p(GetMessagePos());
    myList->ScreenToClient(&p);
    int item=myList->HitTest(CPoint(2,p.y)); if(item<0) return; // something is wrong

    // Create popup menu
    CMenu menu;
    menu.CreatePopupMenu();

    CMenu menu1;
    menu1.LoadMenu(IDR_MENU_FILE_BROWSE);

    CMenu* pop=menu1.GetSubMenu(local?1:2);

    // Set acceptable menu IDs
    UINT ids[5]={ ID_LOCALHOST_OPEN,
                  ID_LOCALHOST_COPYTOREMOTEHOST,
                  ID_LOCALHOST_DELETE,
                  ID_LOCALHOST_RENAME,
                  ID_LOCALHOST_CREATENEWDIRECTORY};

    if(!local)
    {
        ids[0]=ID_REMOTEHOST_OPEN;
        ids[1]=ID_REMOTEHOST_COPYTOREMOTEHOST;
        ids[2]=ID_REMOTEHOST_DELETE;
        ids[3]=ID_REMOTEHOST_RENAME;
        ids[4]=ID_REMOTEHOST_CREATENEWDIRECTORY;
    }

    // Create the popup menu
    int n=0;
    unsigned int nid;
    CString strMenu;
    for(unsigned int i=0; i<pop->GetMenuItemCount(); i++)
    {
        nid=pop->GetMenuItemID(i);
        if( (nid!=ids[0] && nid!=ids[1] && nid!=ids[2]
            && nid!=ids[3] && nid!=ids[4]) continue;
        pop->GetMenuString(i, strMenu, MF_BYPOSITION);
        int status=pop->GetMenuState(i, MF_BYPOSITION);
        menu.InsertMenu(n, status, nid, strMenu);
        n++;
    }
}

```

```
// Display popup menu
ClientToScreen(&p);
p=CPoint(p.x+(local?25:440),p.y+25);
menu.TrackPopupMenu(TPM_LEFTALIGN, p.x,p.y,this);
pop->DestroyMenu();
menu.DestroyMenu();
menu1.DestroyMenu();
```

```
}
```

```

#if !defined(AFX_FINDREGION_H__2307E526_1F4B_11D3_96A4_00105A21774F__INCLUDED_)
#define AFX_FINDREGION_H__2307E526_1F4B_11D3_96A4_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define CANNOT_COMPARE -1313

#include "Image/Image.h"
// FindRegion.h : header file
//

/////////////////////////////////////////////////////////////////
// FindRegion dialog

class FindRegion : public CDialog
{
// Construction
public:
    bool Initialize(CDC *pDC, Image *pBmp, CRect &ScreenPatternRect, CSize& scroll);
    FindRegion(CWnd* pParent = NULL); // standard constructor
    ~FindRegion();

// Dialog Data
    //{AFX_DATA(FindRegion)
    enum { IDD = IDD_DIALOG_FIND_SIMILAR };
    int f_Percent;
    BOOL f_ReduceNoise;
    CProgressCtrl f_Progress;
    CSliderCtrl f_Speed;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(FindRegion)
    public:
        virtual int DoModal();
    protected:
        virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(FindRegion)
    afx_msg void OnButtonFindBest();
    afx_msg void OnPaint();
    afx_msg void OnButtonFindNext();
    virtual BOOL OnInitDialog();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()

private:
    long f_Pattern_Average;
    long** f_Pattern;
    double f_Correlation;
    CPoint f_ImageStartSearchPoint;
    CSize f_scroll;
    CRect f_ImageFoundRect, f_ScreenFoundRect;
    CRect f_ImagePatternRect, f_ScreenPatternRect;
    CRect f_FoundDisplay, f_PatternDisplay;
    Image* f_pBmp;
    CDC* f_CDC;

    void Draw(UINT code);
    void Erase(UINT code);
    void DisplayPatterns();
    bool AllocatePattern();
    bool DeallocatePattern();
    bool Find(UINT mode);
};

```

```
//{{AFX_INSERT_LOCATION}}  
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.  
  
#endif // !defined(AFX_FINDREGION_H__2307E526_1F4B_11D3_96A4_00105A21774F__INCLUDED_)
```



```

// FindRegion.cpp : implementation file
//

#include "stdafx.h"
#include "DCM.h"
#include "FindRegion.h"

#define PATTERN_RECT    0
#define FOUND_RECT      1
#define FIND_BEST_REGION    0
#define FIND_NEXT_REGION    1

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// FindRegion dialog

FindRegion::FindRegion(CWnd* pParent /*=NULL*/)
    : CDialog(FindRegion::IDD, pParent)
{
    // Some very basic initialization
    int n=theApp.app_ResolutionScaleFactor;
    f_PatternDisplay=CRect( CPoint(175,40), CSize(100*n,100*n) );
    f_FoundDisplay = f_PatternDisplay+CPoint(0,f_PatternDisplay.bottom+10*n);
    f_Pattern=NULL;
    f_Pattern_Average=0;

    //{AFX_DATA_INIT(FindRegion)
    f_Percent = 50;
    f_ReduceNoise = TRUE;
    //}AFX_DATA_INIT
}

FindRegion::~FindRegion() {}

void FindRegion::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{AFX_DATA_MAP(FindRegion)
    DDX_Text(pDX, IDC_DIALOGFIND_PERCENT, f_Percent);
    DDV_MinMaxInt(pDX, f_Percent, 0, 100);
    DDX_Control(pDX, IDC_DIALOGFIND_PROGRESS, f_Progress);
    DDX_Control(pDX, IDC_DIALOGFIND_SLIDER, f_Speed);
    DDX_Check(pDX, IDC_DIALOGFIND_DENOISE, f_ReduceNoise);
    //}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(FindRegion, CDialog)
    //{AFX_MSG_MAP(FindRegion)
    ON_BN_CLICKED(IDC_DIALOGFIND_FIND, OnButtonFindBest)
    ON_WM_PAINT()
    ON_BN_CLICKED(IDC_DIALOGFIND_NEXT, OnButtonFindNext)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// FindRegion message handlers
/*****
*
*   Initialize find parameters
*
*****/
bool FindRegion::Initialize(CDC *pDC, Image *pBmp, CRect &ScreenPatternRect, CSize& scroll)
{
    f_pBmp=pBmp;    if(!f_pBmp) return false;
    f_CDC=pDC;      if(!f_CDC) return false;
    f_scroll=scroll;
    f_ScreenPatternRect=f_ScreenFoundRect=ScreenPatternRect;

```

```

f_ImagePatternRect=f_ImageFoundRect
    =f_pBmp->m_ScreenMap.Screen_to_Image(f_ScreenFoundRect, -f_scroll);

// Force selected region inside the image, if necessary
bool inside=true;
if(f_ImagePatternRect.left<0)
{
    f_ImagePatternRect.left=0;    inside=false;
}
if(f_ImagePatternRect.top<0)
{
    f_ImagePatternRect.top=0;    inside=false;
}
if(f_ImagePatternRect.right>=f_pBmp->GetWidth())
{
    f_ImagePatternRect.right=f_pBmp->GetWidth()-1;    inside=false;
}
if(f_ImagePatternRect.bottom>=f_pBmp->GetHeight())
{
    f_ImagePatternRect.bottom=f_pBmp->GetHeight()-1;    inside=false;
}
if(!inside)
{
    f_ImagePatternRect.NormalizeRect();
    if(f_ImagePatternRect.IsRectEmpty())    return FALSE;
    f_ImageFoundRect=f_ImagePatternRect;
    f_ScreenPatternRect=f_ScreenFoundRect=
        f_pBmp->m_ScreenMap.Image_to_Screen(f_ImageFoundRect,f_scroll);
}
return true;
}

*****
* Find best match
*****
bool FindRegion::Find(UINT mode)
{
    int x,y,dx,dy;
    double temp_corr;
    CPoint best_fit(0,0);
    int rw=f_ImagePatternRect.Width();
    int rh=f_ImagePatternRect.Height();
    if(f_pBmp->GetWidth()-rw<rw || f_pBmp->GetHeight()-rh<rh)
    {
        AfxMessageBox("Search pattern is too large\n"
            "Please cancel search and select smaller pattern",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    }

    // Set coordinate increments
    int accur=f_Speed.GetPos();
    if(accur==f_Speed.GetRangeMax())
    {
        dx=dy=1;    // highest accuracy
    }
    else
    {
        dx=__max(1,rw>>accur);    dy=__max(1,rh>>accur);
    }

    // Perform fit search
    bool found_something=false;
    for(x=f_ImageStartSearchPoint.x; x<f_pBmp->GetWidth()-rw; x += dx)
    {
        if(x%10==0) f_Progress.SetPos(100*x/(f_pBmp->GetWidth()-rw));
        if(abs(x-f_ImagePatternRect.left)<rw)    continue;
        for(y=f_ImageStartSearchPoint.y; y<f_pBmp->GetHeight()-rh; y += dy)
        {
            if(abs(y-f_ImagePatternRect.top)<rh)    continue;
            temp_corr=f_pBmp->Compare(COMPARE_CORRELATION, CPoint(x,y), f_Pattern,
                rw, rh, f_Pattern_Average);
            if(temp_corr==CANNOT_COMPARE)    continue;

```

```

temp_corr = (1.0+temp_corr)/2; // map into [0,1]
if(temp_corr>f_Correlation) // best current
{
    found_something=true;
    f_Correlation=temp_corr;
    best_fit=CPoint(x,y);
    if(mode==FIND_NEXT_REGION)
    {
        f_ImageStartSearchPoint=CPoint(x,y+1);
        goto found;
    }
    temp_corr=f_pBmp->Compare(COMPARE_CORRELATION, CPoint(x,y), f_Pattern,
                             rw, rh, f_Pattern_Average);
}
if(f_ImageStartSearchPoint.y>0) f_ImageStartSearchPoint.y=0;
} // next y
if(f_ImageStartSearchPoint.x>0) f_ImageStartSearchPoint.x=0;
} // next x

if(mode==FIND_NEXT_REGION)
{
    AfxMessageBox("Finished searching the image\n"
                  "Next search will start from the beginning.",
                  MB_ICONINFORMATION);
    f_ImageStartSearchPoint=CPoint(0,0);
    return false;
}

if(!found_something)
{
    AfxMessageBox("Not found");
    return false;
}

found:
    f_Progress.SetPos(0);
    if(best_fit.x==CANNOT_COMPARE && best_fit.y==0) return false;
    Beep(500,100);

    // Display the result
    if(f_ImageFoundRect.EqualRect(f_ImagePatternRect)==FALSE) Erase(FOUND_RECT); // remove prev
ious
    f_ImageFoundRect=CRect(best_fit,f_ImagePatternRect.Size());
    f_ScreenFoundRect=f_pBmp->m_ScreenMap.Image_to_Screen(f_ImageFoundRect,f_scroll);
    Draw(FOUND_RECT);
    return true;
}

/*****
*
* Find Button click
*
*****/

void FindRegion::OnButtonFindBest()
{
    f_ImageStartSearchPoint=CPoint(0,0);
    f_Correlation=0.0;
    if(!Find(FIND_BEST_REGION)) return;
    DisplayPatterns();
    f_ImageStartSearchPoint=CPoint(0,0);
}

void FindRegion::OnButtonFindNext()
{
    UpdateData(TRUE);
    f_Correlation=f_Percent/100.0;
    if(!Find(FIND_NEXT_REGION)) return;
    DisplayPatterns();
    //f_ImageStartSearchPoint=f_ImageFoundRect.TopLeft()+CSize(1,1);
}

/*****
*
* Display modal dialog, and erase all rectangles after
*
*****/
int FindRegion::DoModal()

```

```
// CustomFileDialog.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "CustomFileDialog.h"
#include <dlgs.h>
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CCustomFileDialog
```

```
IMPLEMENT_DYNAMIC(CCustomFileDialog, CFileDialog)
```

```
BEGIN_MESSAGE_MAP(CCustomFileDialog, CFileDialog)
//{{AFX_MSG_MAP(CCustomFileDialog)
ON_BN_CLICKED(IDC_SELECT_ITEMS, OnSelectButton)
ON_WM_CONTEXTMENU()
//}}AFX_MSG_MAP
ON_COMMAND(ID_HELP, OnHelp)
END_MESSAGE_MAP()
```

```
// Filter string
```

```
CString CCustomFileDialog::szCustomDefFilter(_T("All Files (*.*)|*.*|Directories|.|""));
CString CCustomFileDialog::szCustomDefExt(_T("dcm"));
CString CCustomFileDialog::szCustomDefFileName(_T("This is the initial default file name"));
CString CCustomFileDialog::szCustomTitle(_T("Select File or Directory"));
```

```
CCustomFileDialog::CCustomFileDialog(BOOL bOpenFileDialog, DWORD dwFlags,
    LPCTSTR lpszFilter, // = szCustomDefFilter
    LPCTSTR lpszDefExt, // = szCustomDefExt
    LPCTSTR lpszFileName, // = szCustomDefFileName
    CWnd* pParentWnd) : // = NULL
    CFileDialog(bOpenFileDialog, lpszDefExt, lpszFileName, dwFlags, lpszFilter, pParentWnd)
```

```
{
    m_bMulti = FALSE;
    m_SelectSubdirectories = TRUE;
```

```
// Most of the "customization" of CCustomFileDialog is set by the dwFlags
// passed in to the constructor. Attempts to enable these features after
// constructing the CCustomFileDialog, such as accessing the m_ofn structure
// directly, may cause CCustomFileDialog to not work correctly
```

```
m_szBigBuffer[0]='\0';
m_ofn.lpstrFile = m_szBigBuffer;
```

```
if (dwFlags & OFN_ALLOWMULTISELECT)
{
    m_bMulti = TRUE;
    // MFC only provides a 260 character buffer for lpstrFile
    // This is not sufficient when you may be expecting a large number of files.
    m_ofn.nMaxFile = sizeof(m_szBigBuffer);
    if (lpszFileName != NULL)
        lstrcpyn(m_szBigBuffer, lpszFileName, sizeof(m_szBigBuffer));
}
else
    m_ofn.nMaxFile = _MAX_PATH;
```

```
if (dwFlags & OFN_EXPLORER)
{
    if (dwFlags & OFN_ENABLETEMPLATE)
    {
        // give it a custom title, too.
        SetTitle("Select File or Directory");
    }
}
```

```
else
    if (m_ofn.Flags & OFN_EXPLORER)
    {
        // MFC added it, but we don't want it
        m_ofn.Flags &= ~(OFN_EXPLORER | OFN_SHOWHELP);
    }
}
```

```

    }

    // FILEOPENORD & MULTIFILEOPENORD can be found from the Infoviewer
    // at Samples -> MFC Samples -> General MFC Samples -> CLIPART -> COMMDLG.RC
    // These are the customized versions
    if (m_bMulti)
        SetTemplate(CUSTOM_MULTIFILEOPENORD, IDD_CUSTOM_FILE_DIALOG);
    else
        SetTemplate(CUSTOM_FILEOPENORD, IDD_CUSTOM_FILE_DIALOG);
}

void CCustomFileDialog::SetTitle(CString title)
{
    CCustomFileDialog::szCustomTitle = title;
    m_ofn.lpstrTitle = CCustomFileDialog::szCustomTitle;
}

void CCustomFileDialog::DoDataExchange(CDataExchange* pDX)
{
    CFileDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CCustomFileDialog)
    DDX_Check(pDX, IDC_CHECK, m_SelectSubdirectories);
    //}}AFX_DATA_MAP
}

BOOL CCustomFileDialog::ReadListViewNames()
{
    // Okay, this is the big hack of the sample, I admit it.
    // With some creative use of the Spy++ utility, you will
    // find that the listview is not actually ID = lst1 as
    // documented in some references, but is actually a child
    // of dlg item ID = lst2

    // WARNING! Although this is a non-intrusive customization,
    // it does rely on unpublished (but easily obtainable)
    // information. The Windows common file dialog box implementation
    // may be subject to change in future versions of the
    // operating systems, and may even be modified by updates of
    // future Microsoft applications. This code could break in such
    // a case. It is intended to be a demonstration of one way of
    // extending the standard functionality of the common dialog boxes.

    CWnd* pWnd = GetParent()->GetDlgItem(lst2);
    if (pWnd == NULL) return FALSE;

    CListCtrl* wndLst1 = (CListCtrl*)(pWnd->GetDlgItem(1));

    UINT nSelected = wndLst1->GetSelectedCount();
    if (!nSelected) return FALSE; // nothing selected

    CString strDirectory = GetFolderPath();
    if (strDirectory.Right(1) != _T("\\"))
    {
        strDirectory += _T("\\");
    }
    CString strItemText;
    // Could this iteration code be cleaner?
    for (int nItem = wndLst1->GetNextItem(-1, LVNI_SELECTED);
        nSelected-- > 0; nItem = wndLst1->GetNextItem(nItem, LVNI_SELECTED))
    {
        strItemText = strDirectory + wndLst1->GetItemText(nItem, 0);
        m_listDisplayNames.AddHead(strItemText);
    }
    return TRUE;
}

BOOL CCustomFileDialog::OnFileNameOK()
{
    ReadListViewNames();
    // CFileDialog's m_ofn.lpstrFile will contain last set of selections!
    return FALSE;
}

```

```

}

void CCustomFileDialog::OnSelectButton()
{
    ReadListViewNames();
    ((CDialog*)GetParent())->EndDialog(IDOK);
}

void CCustomFileDialog::OnContextMenu(CWnd* pWnd, CPoint point)
{
    const DWORD helpIDs[] =
    {
        //IDC_SELECT_ITEMS,    IDC_SELECT_ITEMS + 0x50000,
        0,0
    };
    ::WinHelp(pWnd->m_hWnd, AfxGetApp()->m_pszHelpFilePath,
        HELP_CONTEXTMENU, (DWORD)(LPVOID)helpIDs);
}

void CCustomFileDialog::OnHelp()
{
    // TODO: How do I bring up the main topic as a contextpopup?
    AfxGetApp()->WinHelp(1, HELP_CONTEXTPOPUP);
}

BOOL CCustomFileDialog::OnInitDialog()
{
    CFileDialog::OnInitDialog();

    // Clear all old selections
    m_listDisplayNames.RemoveAll();

    // Let's change some button text
    // Here's one of the big differences from pre MFC 4.0
    // customizations. All of the controls on the Explorer
    // dialog are now on a dialog which is the PARENT of
    // the CFileDialog. That's right; you'll need to do a
    // GetParent() first, before using GetDlgItem().

    // Overall, this removes the edit control and its static
    // text and then moves the filetypes combo and static
    // text up into the same spot.
    CRect rcWndEdit, rcWndStatic;

    CWnd* pWnd = GetParent()->GetDlgItem(edt1);
    pWnd->GetWindowRect(&rcWndEdit);
    GetParent()->ScreenToClient(&rcWndEdit);
    pWnd = GetParent()->GetDlgItem(stc3);
    pWnd->GetWindowRect(&rcWndStatic);
    GetParent()->ScreenToClient(&rcWndStatic);
    // More undocumented but non-implementation functions
    // Work only when supplying a customized template
    HideControl(edt1); // Hide edit control
    HideControl(stc3); // Hide edit control's text

    pWnd = GetParent()->GetDlgItem(cmb1);
    pWnd->SetWindowPos(NULL, rcWndEdit.left, rcWndEdit.top, 0, 0,
        SWP_NOSIZE | SWP_NOZORDER | SWP_NOACTIVATE);
    pWnd = GetParent()->GetDlgItem(stc2);
    pWnd->SetWindowPos(NULL, rcWndStatic.left, rcWndStatic.top, 0, 0,
        SWP_NOSIZE | SWP_NOZORDER | SWP_NOACTIVATE);

    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

/*****
*
* Return selection # index
*
*****/
CString CCustomFileDialog::GetSelectedAt(UINT index)
{
    CString res("");

```

```
if(index>=(UINT)m_listDisplayNames.GetCount()) return res;  
POSITION pos = m_listDisplayNames.FindIndex(index);  
if(!pos) return res;  
res = (CString)m_listDisplayNames.GetAt(pos);  
return res;  
}
```

```

#if !defined(AFX_COLORDIALOG_H__13C30C65_8287_11D2_9596_000000000000__INCLUDED_)
#define AFX_COLORDIALOG_H__13C30C65_8287_11D2_9596_000000000000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// ColorDialog.h : header file
//

/////////////////////////////////////////////////////////////////
// ColorDialog dialog

class ColorDialog : public CDialog
{
// Construction
public:
    ColorDialog(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(ColorDialog)
    enum { IDD = IDD_DIALOG_COLOR };
    CSliderCtrl m_Contrast;
    CSliderCtrl m_Gamma;
    CSliderCtrl m_Brightness;
    int         m_br_value;
    double      m_gamma_value;
    int         m_con_value;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(ColorDialog)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(ColorDialog)
    virtual BOOL OnInitDialog();
    afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CWnd* m_pView;

    //{{AFX_INSERT_LOCATION}}
    // Microsoft Developer Studio will insert additional declarations immediately before the previous
    line.

#endif // !defined(AFX_COLORDIALOG_H__13C30C65_8287_11D2_9596_000000000000__INCLUDED_)

```



```
// ColorDialog.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "DCM.h"
#include "ColorDialog.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// ColorDialog dialog
```

```
ColorDialog::ColorDialog( CWnd* pParent /*=NULL*/)
: CDialog(ColorDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(ColorDialog)
    m_br_value = 0;
    m_gamma_value = 100;
    m_con_value = 0;
    //}}AFX_DATA_INIT
    m_pView=pParent;
}
```

```
void ColorDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(ColorDialog)
    DDX_Control(pDX, IDC_SLIDER_CONTRAST, m_Contrast);
    DDX_Control(pDX, IDC_SLIDER_GAMMA, m_Gamma);
    DDX_Control(pDX, IDC_SLIDER_BRIGHTNESS, m_Brightness);
    DDX_Text(pDX, IDC_SLIDER_BR_NUMBER, m_br_value);
    DDV_MinMaxInt(pDX, m_br_value, -100, 100);
    DDX_Text(pDX, IDC_SLIDER_GAM_NUMBER, m_gamma_value);
    DDV_MinMaxDouble(pDX, m_gamma_value, 0, 600);
    DDX_Text(pDX, IDC_SLIDER_CT_NUMBER, m_con_value);
    DDV_MinMaxInt(pDX, m_con_value, 0, 101);
    //}}AFX_DATA_MAP
    if(pDX->m_bSaveAndValidate)
    {
        m_br_value=m_Brightness.GetPos();
        m_gamma_value=m_Gamma.GetPos();
        m_con_value=m_Contrast.GetPos();
    }
    else
    {
        m_Brightness.SetPos(m_br_value);
        m_Gamma.SetPos((int)m_gamma_value);
        m_Contrast.SetPos(m_con_value);
    }
}
```

```
BEGIN_MESSAGE_MAP(ColorDialog, CDialog)
    //{{AFX_MSG_MAP(ColorDialog)
    ON_WM_HSCROLL()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// ColorDialog message handlers
```

```
BOOL ColorDialog::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_Brightness.SetRange(-100,100,TRUE);
    m_Brightness.SetTicFreq(20);
    m_Brightness.SetLineSize(1);
    m_Brightness.SetPageSize(10);
}
```

```

    m_Brightness.SetPos(m_br_value);

    m_Gamma.SetRange(0,600,TRUE);
    m_Gamma.SetTicFreq(100);
    m_Gamma.SetLineSize(1);
    m_Gamma.SetPageSize(10);
    m_Gamma.SetPos((int)m_gamma_value);
    CString value;
    value.Format("%3.1f",m_gamma_value/100.0);
    GetDlgItem(IDC_SLIDER_GAM_NUMBER)->SetWindowText(value);

    m_Contrast.SetRange(0,100,TRUE);
    m_Contrast.SetTicFreq(10);
    m_Contrast.SetLineSize(1);
    m_Contrast.SetPageSize(10);
    m_Contrast.SetPos(m_con_value);

    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void ColorDialog::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    CString value;
    int nControl = pScrollBar->GetDlgCtrlID();
    CSliderCtrl* pControl = (CSliderCtrl*) GetDlgItem(nControl);

    switch (nControl)
    {
        case IDC_SLIDER_BRIGHTNESS:
            ASSERT(pControl != NULL);
            m_br_value = pControl->GetPos();
            value.Format("%d",m_br_value);
            GetDlgItem(IDC_SLIDER_BR_NUMBER)->SetWindowText(value);
            break;

        case IDC_SLIDER_CONTRAST:
            ASSERT(pControl != NULL);
            m_con_value = pControl->GetPos();
            value.Format("%d",m_con_value);
            GetDlgItem(IDC_SLIDER_CT_NUMBER)->SetWindowText(value);
            break;

        case IDC_SLIDER_GAMMA:
            ASSERT(pControl != NULL);
            m_gamma_value = pControl->GetPos();
            value.Format("%3.1f",m_gamma_value/100.0);
            GetDlgItem(IDC_SLIDER_GAM_NUMBER)->SetWindowText(value);
            break;

        default:
            CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
            break;
    }
    /* For Proof later
    if (m_pView)
    {
        m_pView->Invalidate();
        m_pView->UpdateWindow();
    }
    */
    return;
}

```

```

// Compressor.h: interface for the Compressor class.
//
/////////////////////////////////////////////////////////////////
#if !defined(AFX_COMPRESSOR_H__200F3A11_CF87_11D2_9617_00105A21774F__INCLUDED_)
#define AFX_COMPRESSOR_H__200F3A11_CF87_11D2_9617_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "MainFrm.h"

// adding zlib stuff
#if !defined (_WINDOWS)
#define _WINDOWS
#endif
#if !defined (ZLIB_DLL)
#define ZLIB_DLL
#endif
#include "Zlib113\zlib.h"
#define BUFLen 32768

class Compressor
{
public:
    Compressor();
    virtual ~Compressor();
    static bool Z_Compress(CString infile, CString outfile);
    static bool Z_unCompress(CString infile, CString outfile, int max_steps=0);
}

#endif // !defined(AFX_COMPRESSOR_H__200F3A11_CF87_11D2_9617_00105A21774F__INCLUDED_)

```

```

// Compressor.cpp: implementation of the Compressor class.
//
/////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include "DCM.h"
#include "Compressor.h"
#include <io.h>

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

/////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////

Compressor::Compressor()
{

}

Compressor::~Compressor()
{

}

/*
*****
* Uncompress GZipped file
* If "max_steps">0, uncompresses only first max_steps*BUFLen bytes
*****/
bool Compressor::Z_unCompress(CString infile, CString outfile, int max_steps)
{
    long in_size;
    CString message;
    FILE *out;
    gzFile in;
    bool gz_format_error=false;

    // Find compressed file size
    {
        FILE* t_in;
        t_in=fopen(infile,"rb");
        if (t_in == NULL)
        {
            message.Format("Cannot read from file %s",infile);
            if(max_steps>0) AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
            return false;
        }
        in_size = _filelength(_fileno(t_in)); // input file size
        fclose(t_in);
    }

    // Open files for reading/writing
    in=gzopen(infile,"rb");
    if (in == NULL)
    {
        message.Format("Cannot read from file %s",infile);
        if(max_steps>0) AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
        return false;
    }

    out = fopen(outfile, "wb");
    if (out == NULL)
    {
        message.Format("Cannot write to file %s",outfile);
        if(max_steps>0) AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
}

```

```

// Put progress control in the main frame status bar
theApp.ShowProgress(1,"Decompressing file with gzip");

// Perform decompression
static char buf[BUFLen];
long len, tot_len=0, steps=0;
for (;;)
{
    if(max_steps==0) theApp.ShowProgress(min(100,(33*tot_len)/ln_size)); // assume 3:1 compress
ion ratio
    len = gzread(in, buf, BUFLen);
    tot_len += len;
    if (len <= 0) break; // eof
    if ((long)fwrite(buf, 1, len, out) != len && !gz_format_error)
    {
        if(max_steps>0) AfxMessageBox("Possible gz format error",MB_ICONEXCLAMATION | MB_OK);
        gz_format_error=true;
    }
    steps++;
    if(max_steps>0 && steps>=max_steps) break;
}

// Clean up
if (fclose(out))
{
    message.Format("Cannot close file %s", outfile);
    if(max_steps>0) AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
    return false;
}
if (gzclose(in) != Z_OK)
{
    //message.Format("Cannot close file %s", infile);
    //if(max_steps>0) AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
    //return false;
}
return true;
}

/*****
*
* Compress GZipped file
* *****/
bool Compressor::Z_Compress(CString infile, CString outfile)
{
    long in_size;
    CString message;
    FILE *in;
    gzFile out;
    bool gz_format_error=false;

    // Open files for reading/writing
    in=fopen(infile,"rb");
    if (in == NULL)
    {
        message.Format("Cannot read from file %s",infile);
        AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    in_size = _filelength(_fileno(in)); // input file size
    out = gzopen(outfile, "wb");
    if (out == NULL)
    {
        message.Format("Cannot write to file %s",outfile);
        AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
        return false;
    }

    // Put progress control in the main frame status bar
    theApp.ShowProgress(1,"Compressing file with gzip");

```

```

// Perform compression
static char buf[BUFLEN];
long len, tot_len=0;
for (;;)
{
    theApp.ShowProgress((100*tot_len)/in_size);
    len=(long)fread(buf, 1, BUFLen, in);
    tot_len += len;
    if (len <= 0) break; // eof
    if (gzwrite(out, buf, len) != len && !gz_format_error)
    {
        AfxMessageBox("Possible gz format error",MB_ICONEXCLAMATION | MB_OK);
        gz_format_error=true;
    }
}

// Clean up
theApp.ShowProgress(0);
if (gzclose(out) != Z_OK)
{
    message.Format("Cannot close file %s", outfile);
    AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
    return false;
}
if (fclose(in))
{
    message.Format("Cannot close file %s", infile);
    AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
    return false;
}
return true;

```

1

```

/*****
 *
 *   Take care of the appropriate file/directory name syntax
 *
 *****/
CString CreateDirectoryDialog::getName()
{
    m_DirName.Remove(' ');
    m_DirName.Remove('/');
    m_DirName.Remove('\\');
    m_DirName = CString("/") + m_DirName;
    return m_DirName;
}

BEGIN_MESSAGE_MAP(CreateDirectoryDialog, CDialog)
    //{AFX_MSG_MAP(CreateDirectoryDialog)
    // NOTE: the ClassWizard will add message map macros here
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////
// CreateDirectoryDialog message handlers

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_CREATEDIRECTORYDIALOG_H__EE93B006_DE6F_11D2_9629_00105A21774F__INCLUDED_)

```



```

}
#ifdef AFX_EDGESDIALOG_H__03D704B4_834D_11D2_9597_000000000000__INCLUDED_
#define AFX_EDGESDIALOG_H__03D704B4_834D_11D2_9597_000000000000__INCLUDED_

}
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// EdgesDialog.h : header file
//

////////////////////////////////////
// EdgesDialog dialog

}
class EdgesDialog : public CDialog
{
// Construction
public:
    BYTE GetThreshold();
    EdgesDialog(int threshold=128,CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(EdgesDialog)
    enum { IDD = IDD_DIALOG_EDGES };
    CSliderCtrl m_SliderEdges;
    BYTE m_Thr_Value;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(EdgesDialog)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
   //{{AFX_MSG(EdgesDialog)
    virtual BOOL OnInitDialog();
    afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
}

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous
// line.

}
#endif // !defined(AFX_EDGESDIALOG_H__03D704B4_834D_11D2_9597_000000000000__INCLUDED_)

```

```

// EdgesDialog.cpp : implementation file
//

#include "stdafx.h"
#include "DCM.h"
#include "EdgesDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// EdgesDialog dialog

EdgesDialog::EdgesDialog(int threshold /*=128*/, CWnd* pParent /*=NULL*/)
: CDialog(EdgesDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(EdgesDialog)
    //}}AFX_DATA_INIT
    m_Thr_Value=(100*threshold)/256;
}

void EdgesDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(EdgesDialog)
    DDX_Control(pDX, IDC_SLIDER_EDGES, m_SliderEdges);
    DDX_Text(pDX, IDC_SLIDER_EG_NUMBER, m_Thr_Value);
    //DDV_MinMaxByte(pDX, m_Thr_Value, 0, 100);
    //}}AFX_DATA_MAP
    if(pDX->m_bSaveAndValidate)
    {
        m_Thr_Value=m_SliderEdges.GetPos();
    }
    else
    {
        m_SliderEdges.SetPos(m_Thr_Value);
    }
}

BEGIN_MESSAGE_MAP(EdgesDialog, CDialog)
    //{{AFX_MSG_MAP(EdgesDialog)
    ON_WM_HSCROLL()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// EdgesDialog message handlers

BOOL EdgesDialog::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_SliderEdges.SetRange(0,100,TRUE);
    m_SliderEdges.SetTicFreq(10);
    m_SliderEdges.SetLineSize(1);
    m_SliderEdges.SetPageSize(10);
    m_SliderEdges.SetPos(m_Thr_Value);
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void EdgesDialog::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    CString value;
    int nControl = pScrollBar->GetDlgCtrlID();
    CSliderCtrl* pControl = (CSliderCtrl*) GetDlgItem(nControl);

    switch (nControl)
    {

```

```

    case IDC_SLIDER_EDGES:
        ASSERT(pControl != NULL);
        m_Thr_Value = pControl->GetPos();
        value.Format("%d", m_Thr_Value);
        GetDlgItem(IDC_SLIDER_EG_NUMBER)->SetWindowText(value);
        break;

    default:
        CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
        break;
}
/* For Proof later
if (m_pView)
{
    m_pView->Invalidate();
    m_pView->UpdateWindow();
}
*/

return;
}

BYTE EdgesDialog::GetThreshold()
{
    int t=m_Thr_Value;
    t=(256*t)/100;  if(t>255) t=255; // convert to 8-bit scale
    return ((BYTE)t);
}

```

```
// Email.h: interface for the Email class.
//
/////////////////////////////////////////////////////////////////
#if !defined(AFX_EMAIL_H_FDCA3A87_F053_11D3_9789_00105A21774F__INCLUDED_)
#define AFX_EMAIL_H_FDCA3A87_F053_11D3_9789_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <map.h>

class Email
{
public:
    bool Send(CString to, CString subject, CString text,
              CString fileattachment, HWND hWnd=NULL);
    Email();
    virtual ~Email();

protected:
    HINSTANCE      m_hlibMAPI;
    LHANDLE        m_lhSession;
    LPMAPISENDMAIL m_MAPISendMail;
    LPMAPILOGON    m_MAPILogon;
    LPMAPILOGOFF   m_MAPILogOff;
    HWND          m_hWndParent;

private:
    bool LogOff();
    bool LogOn(HWND hWnd=NULL);
}

#endif // !defined(AFX_EMAIL_H_FDCA3A87_F053_11D3_9789_00105A21774F__INCLUDED_)
```

```

// Email.cpp: implementation of the Email class.
//
/////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Email.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif

/////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////

Email::Email()
{
    m_hWndParent=0;
    m_lhSession=0;
    m_hlibMAPI=0;
    m_MAPILogon = 0;
    m_MAPILogOff = 0;
    m_MAPISendMail = 0;
}

Email::~Email()
{
}

/* *****
 * Log to the Simple MAPI service
 * ***** */
bool Email::LogOn(HWND hWnd /*=NULL*/)
{
    if(m_lhSession!=0)
    {
        AfxMessageBox("Cannot load email service");
        return false;
    }
    // Load Simple MAPI dll
    m_hlibMAPI = LoadLibrary("MAPI32.DLL"); // 32 bit clients
    if(!m_hlibMAPI)
    {
        AfxMessageBox("Cannot load email service");
        return false;
    }
    m_hWndParent = hWnd;

    // Load MAPI functions
    m_MAPILogon = (LPMAPILOGON)GetProcAddress(m_hlibMAPI, "MAPILogon");
    m_MAPILogOff = (LPMAPILOGOFF)GetProcAddress(m_hlibMAPI, "MAPILogoff");
    m_MAPISendMail = (LPMAPISENDMAIL)GetProcAddress(m_hlibMAPI, "MAPISendMail");

    // Logon to MAPI
    switch(m_MAPILogon((ULONG)m_hWndParent, NULL, NULL,
        MAPI_PASSWORD_UI/* | MAPI_FORCE_DOWNLOAD*/, 0L, &m_lhSession))
    {
    case MAPI_E_INSUFFICIENT_MEMORY:
        AfxMessageBox("Insufficient memory to proceed");
        return false;
    case MAPI_E_LOGIN_FAILURE:
        AfxMessageBox("Failed to log on");
        return false;
    case MAPI_E_TOO_MANY_SESSIONS:
        AfxMessageBox("Failed: Too many email sessions open simultaneously");
        return false;
    case MAPI_E_USER_ABORT:
        AfxMessageBox("Failed: User abort");
        return false;
    }
}

```

```

    case SUCCESS_SUCCESS:
        return true;
    default: // MAPI_E_FAILURE:
        AfxMessageBox("Failed to start email service");
        return false;
    }
    return false; // must never get here
}

/*****
*
*   Log off the Simple MAPI service,
*   clean MAPI memory buffers (if needed)
*
*****/
bool Email::LogOff()
{
    if(m_MAPILogOff(m_lhSession, (ULONG)m_hWndParent, 0L, 0L)!=SUCCESS_SUCCESS)
    {
        AfxMessageBox("Failed to terminate email session");
        return false;
    }
    m_lhSession=0;
    return true;
}

/*****
*
*   Send the message
*
*****/
bool Email::Send(CString to, CString subject, CString text,
                CString fileattachment, HWND hWnd/*=NULL*/)
{
    if(!LogOn(hWnd)) return false;
    const ULONG ulReserved = 0L;

    // Initialize file attachment structure
    MapiFileDesc attachment;
    CString name=fileattachment;
    if(fileattachment!="")
    {
        int n = fileattachment.ReverseFind('/');
        if(n<0) n = fileattachment.ReverseFind('\\');
        if(n>0) name=fileattachment.Mid(n+1);
        attachment.ulReserved=ulReserved;
        attachment.flFlags=0;
        attachment.nPosition=(ULONG)-1;
        attachment.lpszPathName=(char*)(LPCSTR)fileattachment;
        attachment.lpszFileName=(char*)(LPCSTR)name;
        attachment.lpFileType=NULL;
    }

    // Initialize "to" recipient
    CString toAddress=CString("SMTP:")+to;
    MapiRecipDesc recips;
    recips.ulReserved = ulReserved;
    recips.ulRecipClass = MAPI_TO;
    recips.lpszName = (char*)(LPCSTR)to;
    recips.lpszAddress = (char*)(LPCSTR)toAddress;
    recips.ulEIDSize = 0;
    recips.lpEntryID =NULL;

    // Initialize MAPI message for one recipient, with at most one attachment
    MapiMessage message;
    message.ulReserved = ulReserved;
    message.lpszSubject = (char*)(LPCSTR)subject;
    message.lpszNoteText = (char*)(LPCSTR)text;
    message.lpszMessageType = NULL;
    message.lpszDateReceived = NULL;
    message.lpszConversationID = NULL;
    message.flFlags = 0;
    message.lpOriginator = NULL;
    message.nRecipCount = 1;

```

```

message.lpRecips = &recips;
if(fileattachment!="")
{
    message.nFileCount = 1;
    message.lpFiles = &attachment;
}
else
{
    message.nFileCount = 0;
    message.lpFiles = NULL;
}
ULONG err=m_MAPISendMail ( m_lhSession, (ULONG)m_hWndParent,
                           &message, MAPI_DIALOG,0L);

switch(err)
{
case MAPI_E_AMBIGUOUS_RECIPIENT:
    AfxMessageBox("Send failed:\nAmbiguous recipient description");
    break;
case MAPI_E_ATTACHMENT_NOT_FOUND:
    AfxMessageBox("Send failed:\nThe specified attachment was not found");
    break;
case MAPI_E_ATTACHMENT_OPEN_FAILURE:
    AfxMessageBox("Send failed:\nThe specified attachment could not be opened");
    break;
case MAPI_E_BAD_RECIPTYPE:
    AfxMessageBox("Send failed:\nBad recipient type");
    break;
case MAPI_E_INSUFFICIENT_MEMORY:
    AfxMessageBox("Send failed:\nInsufficient memory");
    break;
case MAPI_E_INVALID_RECIPS:
    AfxMessageBox("Send failed:\nInvalid recipient(s)");
    break;
case MAPI_E_LOGIN_FAILURE:
    AfxMessageBox("Send failed:\nFailed to log on successfully");
    break;
case MAPI_E_TEXT_TOO_LARGE:
    AfxMessageBox("Send failed:\nThe text in the message was too large");
    break;
case MAPI_E_TOO_MANY_FILES:
    AfxMessageBox("Send failed:\nThere were too many file attachments");
    break;
case MAPI_E_TOO_MANY_RECIPIENTS:
    AfxMessageBox("Send failed:\nThere were too many recipients");
    break;
case MAPI_E_UNKNOWN_RECIPIENT:
    AfxMessageBox("Send failed:\nUnknown recipient");
    break;
case MAPI_E_USER_ABORT:
    AfxMessageBox("Send failed:\nUser abort");
    break;
case SUCCESS_SUCCESS:
    break;
default: // MAPI_E_FAILURE:
    AfxMessageBox("Send failed");
    break;
}

return (err==SUCCESS_SUCCESS) && LogOff();
}

```

[illegible]



```

#ifdef AFX_FILEBROWSER_H_B5F74D80_CC60_11D2_9614_00105A21774F__INCLUDED_
#define AFX_FILEBROWSER_H_B5F74D80_CC60_11D2_9614_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
//FileBrowser.h : header file
//

#include "afxinet.h"
#include "FTPLoginDialog.h" // Added by ClassView

////////////////////////////////////
// FileBrowser dialog

class FileBrowser : public CDialog
{
// Construction
public:
    bool Initialize(CString temp_dir);
    CString m_RequestedFile;
    FileBrowser(CWnd* pParent = NULL); // standard constructor
    ~FileBrowser(); // destructor

// Dialog Data
    //{AFX_DATA(FileBrowser)
    enum { IDD = IDD_DIALOG_FILE_BROWSE };
    CListCtrl m_List_FTP;
    CComboBox m_DriveList;
    CListCtrl m_List;
    CString m_Directory;
    int m_NumFiles;
    int m_NumFilesTotal;
    CString m_Directory_FTP;
    int m_NumFiles_FTP;
    int m_NumFilesTotal_FTP;
    CString m_INhost;
    CString m_SpeedInfo;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(FileBrowser)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(FileBrowser)
    virtual BOOL OnInitDialog();
    afx_msg void OnDblclkList(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnSelchangeFileBrowseCombo();
    afx_msg void OnColumnclickList(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnButtonFtpGet();
    afx_msg void OnButtonFtpPut();
    afx_msg void OnLocRefresh();
    afx_msg void OnFtpRefresh();
    afx_msg void OnGoFtp();
    afx_msg void OnLocalhostShowDICOMonly();
    afx_msg void OnUpdateLocalhostShowDICOMonly(CCmdUI* pCmdUI);
    afx_msg void OnRemotehostFilterDICOMfiles();
    afx_msg void OnUpdateRemotehostFilterDICOMfiles(CCmdUI* pCmdUI);
    afx_msg void OnRemotehostShowDICOMonly();
    afx_msg void OnUpdateRemotehostShowDICOMonly(CCmdUI* pCmdUI);
    afx_msg void OnLocalhostDelete();
    afx_msg void OnRemotehostDelete();
    afx_msg void OnLocalhostRename();
    afx_msg void OnRemotehostRename();
    afx_msg void OnRclickList(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnLocalhostCreateNewDirectory();

```

```

afx_msg void OnRemotehostCreateNewDirectory();
afx_msg void OnRemotehostOpen();
afx_msg void OnLocalhostOpen();
afx_msg void OnUpdateLocalhostOpen(CCmdUI* pCmdUI);
afx_msg void OnUpdateRemotehostOpen(CCmdUI* pCmdUI);
//}}AFX_MSG
afx_msg LRESULT OnKickIdle(WPARAM, LPARAM);
DECLARE_MESSAGE_MAP()

private:
void OpenFile_or_Directory(bool local, int item);
bool CreateNewDirectory(CListCtrl &myList, bool is_local);
bool RenameFile_or_Directory(CListCtrl & myList, bool is_local);
bool DeleteFile_or_Directory(CListCtrl & myList, bool is_local);
bool m_DCMonly_FTP;
bool FileTransfer(CString ftp_name, CString loc_name, bool to_local, int size=-1);
int FindName(CString fname, CListCtrl & myList);
void DICOM_Filter(CListCtrl & myList, bool ftp);
bool resetFTP();
CString m_ParentDirectory_FTP;
void deleteFTP();
bool resetFTP(CString host, CString logon, CString pwd);
bool m_FilterFTP;
bool m_DCMonly_loc;
CString m_INpassword;
CString m_INlogon;
CFTPConnection* m_pFTPConnection;
CInternetSession m_InSession;
void OpenedFilesListRemove(CString fullname);
bool OpenedFilesListFind(CString fullname);
void OpenedFilesListAdd(CString fullname);
CMap<CString,LPCSTR,int, int> m_Opened;
CString m_TempFile;
CString m_ParentDirectory;
void SetItemImageState(int nitem, int state, CListCtrl & myList);
int GetItemImageState(int nitem, CListCtrl & myList);
CImageList m_ImageList;
void SortByColumn(int col, CListCtrl & myList);
int FindFiles(CListCtrl & myList, bool ftp, CString directory="");
int GetSelectedPosition(bool local);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FILEBROWSER_H__B5F74D80_CC60_11D2_9614_00105A21774F__INCLUDED_)

```

```

        filter.Format("%s %s='%s' ", and, fName, fValue);
    }
    m_strFilter += filter;
}

void ODBCTableSet::AndFilter(CString fName, DateTimeSegment &dValue,
                             const BYTE dFormat)
{
    CString filter("");
    CString and = (m_strFilter == "" ? "" : " AND");
    if(dFormat==DateTime::DateFormat)    // Date
    {
        int nStart = dValue.GetStart().GetNumericDate();
        int nEnd   = dValue.GetEnd().GetNumericDate();

        if(nStart==nEnd)    // exact date
        {
            if(nStart<0)    return;
            filter.Format("%s %s=%d ", and, fName, nStart);
            m_strFilter += filter;
            return;
        }

        // We have interval
        if(nStart<0)    nStart=0;
        if(nEnd<0)
        {
            filter.Format("%s %s>=%d ", and, fName, nStart);
        }
        else
        {
            filter.Format("%s %s BETWEEN %d AND %d ", and, fName,
                           nStart, nEnd);
        }
        m_strFilter += filter;
        return;
    }
    else if(dFormat==DateTime::TimeFormat)    // Time
    {
        double nStart = dValue.GetStart().GetNumericTime();
        double nEnd   = dValue.GetEnd().GetNumericTime();

        if(nStart==nEnd)    // exact date
        {
            if(nStart<0)    return;
            filter.Format("%s %s=%f ", and, fName, nStart);
            m_strFilter += filter;
            return;
        }

        // We have interval
        if(nStart<0)    nStart=0;
        if(nEnd<0)
        {
            filter.Format("%s %s>=%f ", and, fName, nStart);
        }
        else
        {
            filter.Format("%s %s BETWEEN %f AND %f ", and, fName,
                           nStart, nEnd);
        }
        m_strFilter += filter;
        return;
    }
    return;
}

/*****
*
*   Add a record to the database, if it did not exist,
*   or update record empty fields, if the record already existed
*
*****/
bool ODBCTableSet::AddOrUpdate(DICOMRecord &dr)
{

```

```

TRY
{
    if(IsOpen())        Close();
    if(!Open(CRecordset::dynaset,(LPCSTR)GetDefaultQuery(dr)))    return false;
    if(!CanUpdate())    {    Close(); return false; }
    if(IsEOF()) // need to insert new record
    {
        AddNew();
        if(!SetFromDICOMRecord(dr)) {    Close(); return false; }
        if(!Update())    {    Close(); return false; }
    }
    else        // need to update empty fields in existing record
    {
        Edit();
        UpdateFromDICOMRecord(dr);
        Update();
    }
    Close();
    return true;
}
CATCH(CException, e)
{
    #ifdef _DEBUG
    e->ReportError();
    #endif
    return false;
}
END_CATCH;
return false;
}
}
// *****
// ***** Write data into DICOMObject via DICOMRecord *****
// *****
void ODBCTableSet::WriteIntoDICOMObject(DICOMObject &dob, DICOMObject *dob_mask)
{
    DICOMRecord dr;
    WriteIntoDICOMRecord(dr);
    dr.WriteIntoDICOMObject(dob, dob_mask);
}
// *****
// ***** ODBC PatientSet *****
// *****
IMPLEMENT_DYNAMIC(ODBCPatientSet, CRecordset)

ODBCPatientSet::ODBCPatientSet(CDatabase* pdb)
: ODBCTableSet(theApp.app_DataBase.GetCDatabasePtr())
{
    //{AFX_FIELD_INIT(ODBCPatientSet)
    ClearSet();
    m_nFields = 4;
    //}AFX_FIELD_INIT
}

// *****
// ***** Defining SQL parameters *****
// *****
CString ODBCPatientSet::GetDefaultQuery(DICOMRecord& dr)
{
    CString query;
    query.Format("SELECT * FROM [Patient] WHERE PatientID='%s'",
        ::Trim(dr.GetPatientID()));
    return query;
}

CString ODBCPatientSet::GetDefaultSQL()
{
    return T("[Patient]");
}

```

```

}

void ODBCPatientSet::SetFindFilter(DICOMRecord &dr)
{
    m_strFilter = "";

    // Add whatever is known from DICOMRecord. If the primary key
    // is known for a table, do not use other table fields
    AndFilter("PatientID", dr.GetPatientID());
    if(!::IsUniqueString(dr.GetPatientID()))
    {
        AndFilter("PatientName", dr.GetPatientName());
        AndFilter("PBirthDate", dr.GetPBirthDate(), DateTime::DateFormat);
        AndFilter("PBirthTime", dr.GetPBirthTime(), DateTime::TimeFormat);
    }
}

void ODBCPatientSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{AFX_FIELD_MAP(ODBCPatientSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[PatientID]"), m_PatientID);
    RFX_Text(pFX, _T("[PatientName]"), m_PatientName);
    RFX_Double(pFX, _T("[PBirthTime]"), m_PBirthTime);
    RFX_Long(pFX, _T("[PBirthDate]"), m_PBirthDate);
    //}AFX_FIELD_MAP
}

/*****
*
* Clear patient recordset data
*
*****/
void ODBCPatientSet::ClearSet()
{
    m_PatientID = _T("");    m_PatientName = _T("");
    m_PBirthTime = -1.0;    m_PBirthDate = -1;
}

/*****
*
* Exchanging data with DICOMRecord
*
*****/
void ODBCPatientSet::UpdateFromDICOMRecord(DICOMRecord &dr)
{
    char* s;
    if(m_PatientName=="")
    {
        s=dr.GetPatientName();
        if(!::IsEmptyString(s)) m_PatientName = ::Trim(s);
    }
    if(m_PBirthDate <= 0)
    {
        int nDate = dr.GetPBirthDate().GetStart().GetNumericDate();
        if(nDate > 0) m_PBirthDate = nDate;
    }
    if(m_PBirthTime < 0)
    {
        double dTime = dr.GetPBirthDate().GetStart().GetNumericTime();
        if(dTime >= 0) m_PBirthTime=dTime;
    }
}

bool ODBCPatientSet::SetFromDICOMRecord(DICOMRecord &dr)
{
    if(!::IsUniqueString(dr.GetPatientID()))    return false;
    ClearSet();
    UpdateFromDICOMRecord(dr);
    m_PatientID = ::Trim(dr.GetPatientID());
    return true;
}

void ODBCPatientSet::WriteIntoDICOMRecord(DICOMRecord &dr)

```

```

{
    dr.SetRecord((char*) (LPCSTR)m_PatientID, (char*) (LPCSTR)m_PatientName,
        m_PBirthDate, m_PBirthTime, NULL,
        NULL, NULL,
        NULL, -1, -1.0,
        NULL, NULL,
        NULL, NULL,
        NULL, NULL);
}

////////////////////////////////////
//
//  ODBCStudySet
//
////////////////////////////////////
IMPLEMENT_DYNAMIC(ODBCStudySet, CRecordset)

ODBCStudySet::ODBCStudySet(CDatabase* pdb)
    : ODBCTableSet(theApp.app_DataBase.GetCDatabasePtr())
{
    //{AFX_FIELD_INIT(ODBCStudySet)
    ClearSet();
    m_nFields = 7;
    //}AFX_FIELD_INIT
}

void ODBCStudySet::ClearSet()
{
    m_PatientID = _T("");          m_StudyInstUID = _T("");
    m_StudyID = _T("");           m_AccessionNumber = _T("");
    m_StudyTime = -1.0;           m_StudyDate = -1;
    m_StudyImagesNum = _T("");
}

*****
*
*   Defining SQL parameters
*
*****/
CString ODBCStudySet::GetDefaultQuery(DICOMRecord &dr)
{
    CString query;
    query.Format("SELECT * FROM [Study] WHERE StudyInstUID='%s'",
        ::Trim(dr.GetStudyInstUID()));
    return query;
}

CString ODBCStudySet::GetDefaultSQL()
{
    return _T("[Study]");
}

void ODBCStudySet::SetFindFilter(DICOMRecord &dr)
{
    m_strFilter = "";

    // Add whatever is known from DICOMRecord. If the primary key
    // is known for a table, do not use other table fields
    AndFilter("StudyInstUID", dr.GetStudyInstUID());
    if(!::IsUniqueString(dr.GetStudyInstUID()))
    {
        AndFilter("PatientID", dr.GetPatientID());
        AndFilter("StudyID", dr.GetStudyID());
        AndFilter("StudyImagesNum", dr.GetStudyImagesNum());
        AndFilter("AccessionNumber", dr.GetAccessionNumber());
        AndFilter("StudyDate", dr.GetStudyDate(), DateTime::DateFormat);
        AndFilter("StudyTime", dr.GetStudyTime(), DateTime::TimeFormat);
    }
}

void ODBCStudySet::DoFieldExchange(CFieldExchange* pFX)
{
    //{AFX_FIELD_MAP(ODBCStudySet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[PatientID]"), m_PatientID);
    RFX_Text(pFX, _T("[StudyInstUID]"), m_StudyInstUID);
    RFX_Text(pFX, _T("[StudyID]"), m_StudyID);
    RFX_Text(pFX, _T("[AccessionNumber]"), m_AccessionNumber);
    RFX_Double(pFX, _T("[StudyTime]"), m_StudyTime);
}

```

```

RFX_Long(pFX, _T("[StudyDate]"), m_StudyDate);
RFX_Text(pFX, _T("[StudyImagesNum]"), m_StudyImagesNum);
//}}AFX_FIELD_MAP

```

```

/*****

```

```

*   Exchanging data with DICOMRecord

```

```

*****/

```

```

void ODBCStudySet::UpdateFromDICOMRecord(DICOMRecord &dr)

```

```

{
    char* s;
    if(m_PatientID=="")
    {
        s=dr.GetPatientID();
        if(!::IsEmptyString(s)) m_PatientID = ::Trim(s);
    }
    if(m_StudyID=="")
    {
        s=dr.GetStudyID();
        if(!::IsEmptyString(s)) m_StudyID = ::Trim(s);
    }
    if(m_AccessionNumber=="")
    {
        s = dr.GetAccessionNumber();
        if(!::IsEmptyString(s)) m_AccessionNumber = ::Trim(s);
    }
    if(m_StudyImagesNum=="")
    {
        s = dr.GetStudyImagesNum();
        if(!::IsEmptyString(s)) m_StudyImagesNum = ::Trim(s);
    }
    if(m_StudyDate <= 0)
    {
        int nDate = dr.GetStudyDate().GetStart().GetNumericDate();
        if(nDate > 0) m_StudyDate = nDate;
    }
    if(m_StudyTime < 0)
    {
        double dTime = dr.GetStudyTime().GetStart().GetNumericTime();
        if(dTime >= 0) m_StudyTime=dTime;
    }
}

```

```

bool ODBCStudySet::SetFromDICOMRecord(DICOMRecord &dr)

```

```

{
    if(!::IsUniqueString(dr.GetStudyInstUID())) return false;
    ClearSet();
    UpdateFromDICOMRecord(dr);
    m_StudyInstUID = ::Trim(dr.GetStudyInstUID());
    return true;
}

```

```

void ODBCStudySet::WriteIntoDICOMRecord(DICOMRecord &dr)

```

```

{
    dr.SetRecord((char*) (LPCSTR)m_PatientID, NULL,
        -1, -1.0, (char*) (LPCSTR)m_StudyInstUID,
        (char*) (LPCSTR)m_StudyID, (char*) (LPCSTR)m_AccessionNumber,
        (char*) (LPCSTR)m_StudyImagesNum, m_StudyDate, m_StudyTime,
        NULL, NULL,
        NULL, NULL,
        NULL, NULL);
}

```

```

////////////////////////////////////
//
//   ODBCSeriesSet
//
////////////////////////////////////
IMPLEMENT_DYNAMIC(ODBCSeriesSet, CRecordset)

```

```

ODBCSeriesSet::ODBCSeriesSet(CDatabase* pdb)
: ODBCTableSet(theApp.app_DataBase.GetCDatabasePtr())
{
    //{{AFX_FIELD_INIT(ODBCSeriesSet)

```

```

    ClearSet();
    m_nFields = 4;
    //}}AFX_FIELD_INIT
}

void ODBCSeriesSet::ClearSet()
{
    m_StudyInstUID = _T("");    m_SeriesInstUID = _T("");
    m_Modality = _T("");    m_SeriesNum = _T("");
}

/*****
*
*   Defining SQL parameters
*
*****/
CString ODBCSeriesSet::GetDefaultQuery(DICOMRecord &dr)
{
    CString query;
    query.Format("SELECT * FROM [Series] WHERE SeriesInstUID='%s'",
        ::Trim(dr.GetSeriesInstUID()));
    return query;
}

CString ODBCSeriesSet::GetDefaultSQL()
{
    return _T("[Series]");
}

void ODBCSeriesSet::SetFindFilter(DICOMRecord &dr)
{
    m_strFilter = "";

    // Add whatever is known from DICOMRecord. If the primary key
    // is known for a table, do not use other table fields
    AndFilter("SeriesInstUID", dr.GetSeriesInstUID());
    if(!::IsUniqueString(dr.GetSeriesInstUID()))
    {
        AndFilter("StudyInstUID", dr.GetStudyInstUID());
        AndFilter("Modality", dr.GetModality());
        AndFilter("SeriesNum", dr.GetSeriesNum());
    }
}

void ODBCSeriesSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{{AFX_FIELD_MAP(ODBCSeriesSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[StudyInstUID]"), m_StudyInstUID);
    RFX_Text(pFX, _T("[SeriesInstUID]"), m_SeriesInstUID);
    RFX_Text(pFX, _T("[Modality]"), m_Modality);
    RFX_Text(pFX, _T("[SeriesNum]"), m_SeriesNum);
    //}}AFX_FIELD_MAP
}

/*****
*
*   Exchanging data with DICOMRecord
*
*****/
void ODBCSeriesSet::UpdateFromDICOMRecord(DICOMRecord &dr)
{
    char* s;
    //static UINT alt=0;
    if(m_StudyInstUID=="")
    {
        s = dr.GetStudyInstUID();
        if(!::IsEmptyString(s)) m_StudyInstUID = ::Trim(s);
    }
    /*
    else
    {
        // Consistency check
        CString prkey = ::Trim(dr.GetSeriesInstUID());
        if( m_SeriesInstUID == prkey &&
            m_StudyInstUID != dr.GetStudyInstUID())
        {

```



```

        alt++;
        m_SeriesInstUID.Format("%s.%d", prkey, alt);
        sprintf(dr.GetSeriesInstUID(), "%s", m_SeriesInstUID.Left(64));
    }
}
*/
if(m_Modality=="")
{
    s = dr.GetModality();
    if(!::IsEmptyString(s)) m_Modality = ::Trim(s);
}
if(m_SeriesNum=="")
{
    s = dr.GetSeriesNum();
    if(!::IsEmptyString(s)) m_SeriesNum = ::Trim(s);
}
}

bool ODBCSeriesSet::SetFromDICOMRecord(DICOMRecord &dr)
{
    if(!::IsUniqueString(dr.GetSeriesInstUID()))    return false;
    ClearSet();
    UpdateFromDICOMRecord(dr);
    m_SeriesInstUID = ::Trim(dr.GetSeriesInstUID());
    return true;
}

void ODBCSeriesSet::WriteIntoDICOMRecord(DICOMRecord &dr)
{
    dr.SetRecord(NULL, NULL,
        -1, -1.0, (char*) (LPCSTR)m_StudyInstUID,
        NULL, NULL,
        NULL, -1, -1.0,
        (char*) (LPCSTR)m_SeriesInstUID, (char*) (LPCSTR)m_Modality,
        (char*) (LPCSTR)m_SeriesNum, NULL,
        NULL, NULL);
}

//=====
// ODBCImageSet
//=====
IMPLEMENT_DYNAMIC(ODBCImageSet, CRecordset)

ODBCImageSet::ODBCImageSet(CDatabase* pdb)
: ODBCTableSet(theApp.app_DataBase.GetCDatabasePtr())
{
    //{AFX_FIELD_INIT(ODBCImageSet)
    ClearSet();
    m_nFields = 4;
    //}AFX_FIELD_INIT
}

void ODBCImageSet::ClearSet()
{
    m_SeriesInstUID = _T("");    m_SOPInstUID = _T("");
    m_ImageNum = _T("");    m_Filename = _T("");
}

/*****
*
*   Defining SQL parameters
*
*****/
CString ODBCImageSet::GetDefaultQuery(DICOMRecord &dr)
{
    CString query;
    query.Format("SELECT * FROM [Image] WHERE SOPInstUID='%s'",
        ::Trim(dr.GetSOPInstUID()));
    return query;
}

CString ODBCImageSet::GetDefaultSQL()
{
    return _T("[Image]");
}

void ODBCImageSet::SetFindFilter(DICOMRecord &dr)
{
    m_strFilter = "";
}

```

```

// Add whatever is known from DICOMRecord. If the primary key
// is known for a table, do not use other table fields
AndFilter("SOPInstUID", dr.GetSOPInstUID());
if(!::IsUniqueString(dr.GetSOPInstUID()))
{
    AndFilter("SeriesInstUID", dr.GetSeriesInstUID());
    AndFilter("ImageNum", dr.GetImageNum());
    AndFilter("Filename", dr.GetFileName());
}
}

void ODBCImageSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{AFX_FIELD_MAP(ODBCImageSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[SeriesInstUID]"), m_SeriesInstUID);
    RFX_Text(pFX, _T("[SOPInstUID]"), m_SOPInstUID);
    RFX_Text(pFX, _T("[ImageNum]"), m_ImageNum);
    RFX_Text(pFX, _T("[Filename]"), m_Filename);
    //}AFX_FIELD_MAP
}

/*****
*
*   Exchanging data with DICOMRecord
*
*****/
void ODBCImageSet::UpdateFromDICOMRecord(DICOMRecord &dr)
{
    char* s;
    if(m_SeriesInstUID=="")
    {
        s = dr.GetSeriesInstUID();
        if(!::IsEmptyString(s)) m_SeriesInstUID = ::Trim(s);
    }
    if(m_ImageNum=="")
    {
        s = dr.GetImageNum();
        if(!::IsEmptyString(s)) m_ImageNum = ::Trim(s);
    }
    if(m_Filename=="")
    {
        s = dr.GetFileName();
        if(!::IsEmptyString(s)) m_Filename = ::Trim(s);
    }
}

bool ODBCImageSet::SetFromDICOMRecord(DICOMRecord &dr)
{
    if(!::IsUniqueString(dr.GetSOPInstUID())) return false;
    ClearSet();
    UpdateFromDICOMRecord(dr);
    m_SOPInstUID = ::Trim(dr.GetSOPInstUID());
    return true;
}

void ODBCImageSet::WriteIntoDICOMRecord(DICOMRecord &dr)
{
    dr.SetRecord(NULL, NULL,
        -1, -1.0, NULL,
        NULL, NULL,
        NULL, -1, -1.0,
        (char*)(LPCSTR)m_SeriesInstUID, NULL,
        NULL, (char*)(LPCSTR)m_SOPInstUID,
        (char*)(LPCSTR)m_ImageNum, (char*)(LPCSTR)m_Filename);
}

////////////////////////////////////
// ODBC4Set

IMPLEMENT_DYNAMIC(ODBC4Set, CRecordset)

ODBC4Set::ODBC4Set(CDatabase* pDb)
: ODBCTableSet(theApp.app_DataBase.GetCDatabasePtr())
{

```

```

//{{AFX_FIELD_INIT(ODBC4Set)
ClearSet();
m_nFields = 19;
//}}AFX_FIELD_INIT
}

void ODBC4Set::ClearSet()
{
    m_SeriesInstUID = _T("");    m_SOPInstUID = _T("");
    m_ImageNum = _T("");        m_Filename = _T("");
    m_PatientID = _T("");       m_PatientName = _T("");
    m_PBirthTime = -1.0;        m_PBirthDate = -1;
    m_StudyInstUID = _T("");    m_SeriesInstUID2 = _T("");
    m_Modality = _T("");        m_SeriesNum = _T("");
    m_PatientID2 = _T("");       m_StudyInstUID2 = _T("");
    m_StudyID = _T("");         m_AccessionNumber = _T("");
    m_StudyTime = -1.0;         m_StudyDate = -1;
    m_StudyImagesNum = _T("");
}

CString ODBC4Set::GetDefaultSQL()
{
    return _T("[Image],[Patient],[Series],[Study]");
}

/*****
*
*   Redefines pure virtual from the base class. Unused
*
*****/
CString ODBC4Set::GetDefaultQuery(DICOMRecord &dr)
{
    m_strFilter = "[Patient].[PatientID]=[Study].[PatientID] AND "
                  "[Study].[StudyInstUID]=[Series].[StudyInstUID] AND "
                  "[Series].[SeriesInstUID]=[Image].[SeriesInstUID] ";
    return m_strFilter;
}

void ODBC4Set::DoFieldExchange(CFieldExchange* pFX)
{
    //{{{AFX_FIELD_MAP(ODBC4Set)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[Image].[SeriesInstUID]"), m_SeriesInstUID);
    RFX_Text(pFX, _T("[SOPInstUID]"), m_SOPInstUID);
    RFX_Text(pFX, _T("[ImageNum]"), m_ImageNum);
    RFX_Text(pFX, _T("[Filename]"), m_Filename);
    RFX_Text(pFX, _T("[Patient].[PatientID]"), m_PatientID);
    RFX_Text(pFX, _T("[PatientName]"), m_PatientName);
    RFX_Double(pFX, _T("[PBirthTime]"), m_PBirthTime);
    RFX_Long(pFX, _T("[PBirthDate]"), m_PBirthDate);
    RFX_Text(pFX, _T("[Series].[StudyInstUID]"), m_StudyInstUID);
    RFX_Text(pFX, _T("[Series].[SeriesInstUID]"), m_SeriesInstUID2);
    RFX_Text(pFX, _T("[Modality]"), m_Modality);
    RFX_Text(pFX, _T("[SeriesNum]"), m_SeriesNum);
    RFX_Text(pFX, _T("[Study].[PatientID]"), m_PatientID2);
    RFX_Text(pFX, _T("[Study].[StudyInstUID]"), m_StudyInstUID2);
    RFX_Text(pFX, _T("[StudyID]"), m_StudyID);
    RFX_Text(pFX, _T("[AccessionNumber]"), m_AccessionNumber);
    RFX_Double(pFX, _T("[StudyTime]"), m_StudyTime);
    RFX_Long(pFX, _T("[StudyDate]"), m_StudyDate);
    RFX_Text(pFX, _T("[StudyImagesNum]"), m_StudyImagesNum);
    //}}}AFX_FIELD_MAP
}

/*****
*
*   Set complete WHERE find filter
*
*****/
void ODBC4Set::SetFindFilter(DICOMRecord &dr)
{
    // Relational constraint
    m_strFilter = "[Patient].[PatientID]=[Study].[PatientID] AND "
                  "[Study].[StudyInstUID]=[Series].[StudyInstUID] AND "

```

```

        "[Series].[SeriesInstUID]=[Image].[SeriesInstUID] ";

// Add whatever is known from DICOMRecord. If the primary key
// is known for a table, do not use other table fields
AndFilter("[Patient].[PatientID]", dr.GetPatientID());
if(!::IsUniqueString(dr.GetPatientID()))
{
    AndFilter("PatientName", dr.GetPatientName());
    AndFilter("PBirthDate", dr.GetPBirthDate(), DateTime::DateFormat);
    AndFilter("PBirthTime", dr.GetPBirthTime(), DateTime::TimeFormat);
}

AndFilter("[Study].[StudyInstUID]", dr.GetStudyInstUID());
if(!::IsUniqueString(dr.GetStudyInstUID()))
{
    AndFilter("StudyID", dr.GetStudyID());
    AndFilter("StudyImagesNum", dr.GetStudyImagesNum());
    AndFilter("AccessionNumber", dr.GetAccessionNumber());
    AndFilter("StudyDate", dr.GetStudyDate(), DateTime::DateFormat);
    AndFilter("StudyTime", dr.GetStudyTime(), DateTime::TimeFormat);
}

AndFilter("[Series].[SeriesInstUID]", dr.GetSeriesInstUID());
if(!::IsUniqueString(dr.GetSeriesInstUID()))
{
    AndFilter("Modality", dr.GetModality());
    AndFilter("SeriesNum", dr.GetSeriesNum());
}

AndFilter("SOPInstUID", dr.GetSOPInstUID());
if(!::IsUniqueString(dr.GetSOPInstUID()))
{
    AndFilter("ImageNum", dr.GetImageNum());
    AndFilter("Filename", dr.GetFileName());
}

*****
*
* Put data into DICOM record
*
*****/
void ODBC4Set::WriteIntoDICOMRecord(DICOMRecord &dr)
{
    dr.SetRecord((char*)(LPCSTR)m_PatientID, (char*)(LPCSTR)m_PatientName,
        m_PBirthDate, m_PBirthTime, (char*)(LPCSTR)m_StudyInstUID,
        (char*)(LPCSTR)m_StudyID, (char*)(LPCSTR)m_AccessionNumber,
        (char*)(LPCSTR)m_StudyImagesNum, m_StudyDate, m_StudyTime,
        (char*)(LPCSTR)m_SeriesInstUID, (char*)(LPCSTR)m_Modality,
        (char*)(LPCSTR)m_SeriesNum, (char*)(LPCSTR)m_SOPInstUID,
        (char*)(LPCSTR)m_ImageNum, (char*)(LPCSTR)m_Filename);
}

/*****
*
* Make sure that several different records on one level
* cannot share records on lower level
*
*****/
bool ODBC4Set::HasAliasRecords(DICOMRecord &dr)
{
    CString p=::Trim(dr.GetPatientID());
    CString st=::Trim(dr.GetStudyInstUID());
    CString sr=::Trim(dr.GetSeriesInstUID());
    CString im=::Trim(dr.GetSOPInstUID());
    CString cons;
    cons.Format(
        " AND ( ( [Patient].[PatientID]<>'%s' AND [Study].[StudyInstUID]='%s' ) OR "
        "( [Study].[StudyInstUID]<>'%s' AND [Series].[SeriesInstUID]='%s' ) OR "
        "( [Series].[SeriesInstUID]<>'%s' AND [Image].[SOPInstUID]='%s' ) )", p, st,
        st, sr, sr, im);
    m_strFilter = "[Patient].[PatientID]=[Study].[PatientID] AND "
        "[Study].[StudyInstUID]=[Series].[StudyInstUID] AND "
        "[Series].[SeriesInstUID]=[Image].[SeriesInstUID] ";
    m_strFilter += cons;
}

```

```
bool aliased = true;
TRY
{
    Open();
    aliased = (IsEOF()==FALSE);
    Close();
}
CATCH(CException, e)
{
#ifdef DEBUG
    e->ReportError();
#endif
}
END_CATCH;
return aliased;
}
```

```

#if !defined(AFX_QUERYRETRIEVE_H_INCLUDED_)
#define AFX_QUERYRETRIEVE_H_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// QueryRetrieve.h : header file
//

#include "dqrcontrol.h" // Added by ClassView
#include "..\LogFile.h" // Added by ClassView

////////////////////////////////////
// QueryRetrieve dialog

class QueryRetrieve : public CDialog
{
// Construction
public:
    void SwitchAppearance();
    void DoModalless(CWnd* pParent=NULL);
    void OnBeginDrag(CListCtrl* pList, NMHDR* pNMHDR);
    bool InitializeQueryRetrieve(LogFile* ptrClientLog,
                                LogFile* ptrServerLog, DICOMDatabase* ptrDB);
    QueryRetrieve(CWnd* pParent = NULL); // constructor
    ~QueryRetrieve(); // destructor

// Dialog Data
//{{AFX_DATA(QueryRetrieve)
enum { IDD = IDD_DIALOG_QUERY_RETRIEVE };
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(QueryRetrieve)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    {{{AFX_MSG(QueryRetrieve)
    afx_msg void OnQueryRetrieve_AESetup();
    virtual BOOL OnInitDialog();
    afx_msg void OnParametersPriorityHigh();
    afx_msg void OnParametersPriorityLow();
    afx_msg void OnParametersPriorityNormal();
    afx_msg void OnUpdateParametersPriority(CCmdUI* pCmdUI);
    afx_msg void OnQueryRetrieveClientLog();
    afx_msg void OnQueryRetrieveServerLog();
    afx_msg void OnClose();
    virtual void OnOK();
    afx_msg void OnUpdateQueryRetrieveClientLog(CCmdUI* pCmdUI);
    afx_msg void OnUpdateQueryRetrieveServerLog(CCmdUI* pCmdUI);
    afx_msg void OnQueryRetrieveClearAllLogs();
    afx_msg void OnMenuSelect(UINT nItemID, UINT nFlags, HMENU hSysMenu);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg void OnHide();
    afx_msg void OnExit();
    afx_msg void OnActivate(UINT nState, CWnd* pWndOther, BOOL bMinimized);
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnServicesShowRemoteTasks();
    afx_msg void OnUpdateServicesShowRemoteTasks(CCmdUI* pCmdUI);
    afx_msg void OnServicesTaskScheduling();
    afx_msg void OnUpdateServicesTaskScheduling(CCmdUI* pCmdUI);
    }}}AFX_MSG
    DECLARE_MESSAGE_MAP()

private:
    bool qr_UpdateMenu;
    HWND qr_HWND;
    LogFile *qr_ptrClientLog, *qr_ptrServerLog;
    ApplicationEntityList* qr_AEarray;

```

```
DQRControl      qr_DQRCtrlRemote, qr_DQRCtrlLocal;
CWnd*           qr_pDragWnd;
CImageList*     qr_pDragImage;

void            OnCancel();
void            UpdateMenu (CMenu* pMenu);
BOOL            UpdateData( BOOL bSaveAndValidate=TRUE);
CImageList*     CreateDragImageEx(CListCtrl *pList, LPPOINT lpPoint);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_QUERYRETRIEVE_H_INCLUDED_)
```

```
// QueryRetrieve.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "..\Resource.h"
#include "QueryRetrieve.h"
#include "AEOptions_Dialog.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// QueryRetrieve dialog
```

```
QueryRetrieve::QueryRetrieve(CWnd* pParent /*=NULL*/)
: CDialog(QueryRetrieve::IDD, pParent)
```

```
{
    //{{AFX_DATA_INIT(QueryRetrieve)
    //}}AFX_DATA_INIT
    qr_UpdateMenu=false;
    qr_HWND = NULL;
    qr_AEarray = NULL;
    qr_pDragImage = NULL;
    qr_pDragWnd = NULL;
```

```
QueryRetrieve::~QueryRetrieve()
```

```
void QueryRetrieve::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(QueryRetrieve)
    //}}AFX_DATA_MAP
```

```
BEGIN_MESSAGE_MAP(QueryRetrieve, CDialog)
```

```
    //{{AFX_MSG_MAP(QueryRetrieve)
    ON_COMMAND(ID_QUERYRETRIEVE_AESETUP, OnQueryRetrieve_AESetup)
    ON_COMMAND(ID_PARAMETERS_PRIORITY_HIGH, OnParametersPriorityHigh)
    ON_COMMAND(ID_PARAMETERS_PRIORITY_LOW, OnParametersPriorityLow)
    ON_COMMAND(ID_PARAMETERS_PRIORITY_NORMAL, OnParametersPriorityNormal)
    ON_UPDATE_COMMAND_UI(ID_PARAMETERS_PRIORITY_HIGH, OnUpdateParametersPriority)
    ON_COMMAND(ID_QUERYRETRIEVE_CLIENTLOG, OnQueryRetrieveClientLog)
    ON_COMMAND(ID_QUERYRETRIEVE_SERVERLOG, OnQueryRetrieveServerLog)
    ON_WM_CLOSE()
    ON_UPDATE_COMMAND_UI(ID_QUERYRETRIEVE_CLIENTLOG, OnUpdateQueryRetrieveClientLog)
    ON_UPDATE_COMMAND_UI(ID_QUERYRETRIEVE_SERVERLOG, OnUpdateQueryRetrieveServerLog)
    ON_COMMAND(ID_QUERYRETRIEVE_CLEARALLLOGS, OnQueryRetrieveClearAllLogs)
    ON_WM_MENUSELECT()
    ON_WM_LBUTTONDOWN()
    ON_WM_MOUSEMOVE()
    ON_WM_ACTIVATE()
    ON_WM_SYSCOMMAND()
    ON_COMMAND(ID_SERVICES_SHOWREMOTE TASKS, OnServicesShowRemoteTasks)
    ON_UPDATE_COMMAND_UI(ID_SERVICES_SHOWREMOTE TASKS, OnUpdateServicesShowRemoteTasks)
    ON_COMMAND(ID_SERVICES_TASKSCHEDULING, OnServicesTaskScheduling)
    ON_UPDATE_COMMAND_UI(ID_PARAMETERS_PRIORITY_LOW, OnUpdateParametersPriority)
    ON_UPDATE_COMMAND_UI(ID_PARAMETERS_PRIORITY_NORMAL, OnUpdateParametersPriority)
    ON_UPDATE_COMMAND_UI(ID_SERVICES_TASKSCHEDULING, OnUpdateServicesTaskScheduling)
    //}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// QueryRetrieve message handlers
```

```
/*
*****
*/
```



```

*
*   Alter Query/Retrieve modeless dialog
*
*****/
void QueryRetrieve::DoModeless(CWnd* pParent)
{
    if(GetSafeHwnd())
    {
        ShowWindow(SW_SHOWNORMAL);
        return;
    }
    Create(IDD_DIALOG_QUERY_RETRIEVE, pParent);
    ShowWindow(SW_SHOWNORMAL);
    qr_UpdateMenu=true;
}

void QueryRetrieve::OnClose() { OnHide(); }
void QueryRetrieve::OnOK() { OnHide(); }
void QueryRetrieve::OnCancel() { OnHide(); }
void QueryRetrieve::OnHide()
{
    if(!qr_HWND) return;
    ShowWindow(SW_MINIMIZE);
}

void QueryRetrieve::OnExit()
{
    if(!qr_HWND) return;
    CString msg("Do you want to terminate DICOM Query/Retrieve ?\n");
    msg += CString("If Yes, you will have to restart DCM\n");
    msg += CString("to enable Query/Retrieve again.");
    if( AfxMessageBox(msg, MB_ICONQUESTION | MB_YESNO) == IDNO ) return;
    DestroyWindow();
}

void QueryRetrieve::SwitchAppearance()
{
    if(GetSafeHwnd() == NULL) // Display
    {
        DoModeless(AfxGetMainWnd());
    }
    else // Maximize / Minimize
    {
        if(IsIconic()) ShowWindow(SW_RESTORE);
        else ShowWindow(SW_MINIMIZE);
        UpdateWindow(); // Make sure it redraws the window
    }
}

void QueryRetrieve::OnSysCommand(UINT nID, LPARAM lParam)
{
    if(nID == SC_MAXIMIZE || nID == SC_ZOOM) ShowWindow(SW_RESTORE);
    else CDialog::OnSysCommand(nID, lParam);
    UpdateWindow(); // Make sure it redraws the window
}

void QueryRetrieve::OnActivate(UINT nState, CWnd* pWndOther, BOOL bMinimized)
{
    CDialog::OnActivate(nState, pWndOther, bMinimized);
    if(nState == WA_INACTIVE && pWndOther==AfxGetMainWnd())
        ShowWindow(SW_MINIMIZE);
    UpdateWindow(); // Make sure it redraws the window
}

/*****
*
*   Run AE setup
*
*****/
void QueryRetrieve::OnQueryRetrieve_AESetup()
{
    AEOptions_Dialog aeo_dialog;
    int n = aeo_dialog.DoModal(qr_AEarray);
    if(n>=0)
    {

```

```

    qr_DQRCtrlRemote.LoadArchiveList(n);
    qr_DQRCtrlLocal.LoadArchiveList(n);
}

/*****
*
*   Handle menu updates
*
*****/
void QueryRetrieve::OnMenuSelect(UINT nItemID, UINT nFlags, HMENU hSysMenu)
{
    CDialog::OnMenuSelect(nItemID, nFlags, hSysMenu);
    if (qr_UpdateMenu)    UpdateMenu(GetMenu());
    qr_UpdateMenu = false;
}

void QueryRetrieve::UpdateMenu(CMenu *pMenu)
{
    CCmdUI cmdUI;
    if (!pMenu)    return;
    for (UINT n = 0; n < pMenu->GetMenuItemCount(); ++n)
    {
        CMenu* pSubMenu = pMenu->GetSubMenu(n);
        if (pSubMenu)    UpdateMenu(pSubMenu);    // recursive call
        else
        {
            cmdUI.m_nIndexMax = pMenu->GetMenuItemCount();
            for (UINT i = 0; i < cmdUI.m_nIndexMax; ++i)
            {
                cmdUI.m_nIndex = i;
                cmdUI.m_nID = pMenu->GetMenuItemID(i);
                cmdUI.m_pMenu = pMenu;
                cmdUI.DoUpdate(this, FALSE);    // call handler
            }
        }
    }
}

/*****
*
*   Change priority
*
*****/
void QueryRetrieve::OnParametersPriorityHigh()
{
    qr_DQRCtrlRemote.SetPriority(ServiceClass::HighPriority);
    qr_DQRCtrlLocal.SetPriority(ServiceClass::HighPriority);
}

void QueryRetrieve::OnParametersPriorityLow()
{
    qr_DQRCtrlRemote.SetPriority(ServiceClass::LowPriority);
    qr_DQRCtrlLocal.SetPriority(ServiceClass::LowPriority);
}

void QueryRetrieve::OnParametersPriorityNormal()
{
    qr_DQRCtrlRemote.SetPriority(ServiceClass::NormalPriority);
    qr_DQRCtrlLocal.SetPriority(ServiceClass::NormalPriority);
}

void QueryRetrieve::OnUpdateParametersPriority(CCmdUI* pCmdUI)
{
    BYTE pr;
    switch(pCmdUI->m_nID)
    {
        case ID_PARAMETERS_PRIORITY_HIGH:
            pr=ServiceClass::HighPriority;
            break;
        case ID_PARAMETERS_PRIORITY_NORMAL:
            pr=ServiceClass::NormalPriority;
            break;
        default:
            pr=ServiceClass::LowPriority;
            break;
    }
}

```

```

pCmdUI->SetCheck(qr_DQRctrlRemote.GetPriority() == pr);
qr_UpdateMenu=true;
}

/*****
*
*   Start the dialog
*
*****/
BOOL QueryRetrieve::OnInitDialog()
{
    CDialog::OnInitDialog();
    qr_HWND=GetSafeHwnd();
    qr_UpdateMenu=true;
    if(!qr_DQRctrlRemote.DisplayOverControl(IDC_STATIC_REMOTE, this))
        return FALSE;
    if(!qr_DQRctrlLocal.DisplayOverControl(IDC_STATIC_LOCAL, this))
        return FALSE;
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

/*****
*
*   Set log files (client and server)
*
*****/
bool QueryRetrieve::InitializeQueryRetrieve(LogFile* ptrClientLog,
                                           LogFile* ptrServerLog, DICOMDatabase* ptrDB)
{
    // Set parameters
    if(!ptrClientLog || !ptrServerLog || !ptrDB)
    {
        AfxMessageBox("NULL parameters, cannot initialize Query/Retrieve");
        return false;
    }
    qr_ptrClientLog = ptrClientLog;
    qr_ptrServerLog = ptrServerLog;
    if(! qr_DQRctrlRemote.CreateDQRControl(qr_ptrClientLog, ptrDB, false) ||
        ! qr_DQRctrlLocal.CreateDQRControl(qr_ptrClientLog, ptrDB, true))
    {
        return false;
    }
    qr_AEarray = &(ptrDB->db_AElist);
    return true;
}

/*****
*
*   Show log windows
*
*****/
void QueryRetrieve::OnQueryRetrieveClientLog()
{
    if(!qr_ptrClientLog->IsOn())    qr_ptrClientLog->DoModeless("Client Log");
    else                            qr_ptrClientLog->DestroyWindow();
}
void QueryRetrieve::OnQueryRetrieveServerLog()
{
    if(!qr_ptrServerLog->IsOn())    qr_ptrServerLog->DoModeless("Server Log");
    else                            qr_ptrServerLog->DestroyWindow();
}
void QueryRetrieve::OnUpdateQueryRetrieveClientLog(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(qr_ptrClientLog->IsOn());
    qr_UpdateMenu=true;
}
void QueryRetrieve::OnUpdateQueryRetrieveServerLog(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(qr_ptrServerLog->IsOn());
    qr_UpdateMenu=true;
}

```

```

}
void QueryRetrieve::OnQueryRetrieveClearAllLogs()
{
    qr_ptrClientLog->OnClear();
    qr_ptrServerLog->OnClear();
}

/*****
*
*   Thread-safe UpdateData
*
*****/
BOOL QueryRetrieve::UpdateData(BOOL bSaveAndValidate)
{
    if(qr_HWND) return FromHandle(qr_HWND)->UpdateData(bSaveAndValidate);
    else return FALSE;
}

/*****
*
*   Drag and drop support
*
*****/
CImageList* QueryRetrieve::CreateDragImageEx(CListCtrl *pList, LPPOINT lpPoint)
{
    if (!pList || pList->GetSelectedCount() <= 0) return NULL; //No row selected

    CRect rectSingle, rectComplete(0,0,0,0);

    // Determine List Control Client width size
    pList->GetClientRect(rectSingle);
    int nWidth = rectSingle.Width();

    // Start and Stop index in view area
    int nIndex = pList->GetTopIndex() - 1;
    int nBottomIndex = pList->GetTopIndex() + pList->GetCountPerPage() - 1;
    if (nBottomIndex > (pList->GetItemCount() - 1))
        nBottomIndex = pList->GetItemCount() - 1;

    // Determine the size of the drag image (limite for rows visibled and Client width)
    while ((nIndex = pList->GetNextItem(nIndex, LVNI_SELECTED)) != -1)
    {
        if (nIndex > nBottomIndex)
            break;

        pList->GetItemRect(nIndex, rectSingle, LVIR_BOUNDS);

        if (rectSingle.left < 0)
            rectSingle.left = 0;

        if (rectSingle.right > nWidth)
            rectSingle.right = nWidth;

        rectComplete.UnionRect(rectComplete, rectSingle);
    }

    // Minimize drag rectangle width to the size of the first column
    rectComplete.right = rectComplete.left + pList->GetColumnWidth(0);

    CClientDC dcClient(this);
    CDC dcMem;
    CBitmap Bitmap;

    if (!dcMem.CreateCompatibleDC(&dcClient))
        return NULL;

    if (!Bitmap.CreateCompatibleBitmap(&dcClient, rectComplete.Width(), rectComplete.Height()))
        return NULL;

    CBitmap *pOldMemDCBitmap = dcMem.SelectObject(&Bitmap);
    // Use green as mask color
    dcMem.FillSolidRect(0, 0, rectComplete.Width(), rectComplete.Height(), RGB(0,255,0));

```

```

// Paint each DragImage in the DC
nIndex = pList->GetTopIndex() - 1;
while ((nIndex = pList->GetNextItem(nIndex, LVNI_SELECTED)) != -1)
{
    if (nIndex > nBottomIndex)
        break;

    CPoint pt;
    CImageList* pSingleImageList = pList->CreateDragImage(nIndex, &pt);

    if (pSingleImageList)
    {
        pList->GetItemRect(nIndex, rectSingle, LVIR_BOUNDS);
        pSingleImageList->Draw(&dcMem,
                                0,
                                CPoint(rectSingle.left - rectComplete.left,
                                        rectSingle.top - rectComplete.top),
                                ILC_MASK);
        pSingleImageList->DeleteImageList();
        delete pSingleImageList;
    }

    dcMem.SelectObject(pOldMemDCBitmap);
    CImageList* pCompleteImageList = new CImageList;
    pCompleteImageList->Create(rectComplete.Width(), rectComplete.Height(), ILC_COLOR | ILC_MASK,
0, 1);
    pCompleteImageList->Add(&Bitmap, RGB(0, 255, 0)); // Green is used as mask color
    Bitmap.DeleteObject();

    if (lpPoint)
    {
        lpPoint->x = rectComplete.left;
        lpPoint->y = rectComplete.top;
    }

    return pCompleteImageList;
}

void QueryRetrieve::OnBeginDrag(CListCtrl *pList, NMHDR *pNMHDR)
{
    if (!pList || pList->GetSelectedCount() <= 0)    return;    //No row selected

    NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
    POINT pt;
    qr_pDragImage = CreateDragImageEx(pList, &pt);

    if (qr_pDragImage == NULL)    return;

    qr_pDragWnd = pList;
    CPoint ptStart = pNMListView->ptAction;
    ptStart -= pt;
    qr_pDragImage->BeginDrag(0, ptStart);
    qr_pDragImage->DragEnter(GetDesktopWindow(), pNMListView->ptAction);
    SetCapture();
}

void QueryRetrieve::OnLButtonUp(UINT nFlags, CPoint point)
{
    if (qr_pDragImage && qr_pDragWnd) // In Drag&Drop mode ?
    {
        ::ReleaseCapture();
        qr_pDragImage->DragLeave(GetDesktopWindow());
        qr_pDragImage->EndDrag();

        CPoint pt(point);
        ClientToScreen(&pt);
        CWnd* pDropWnd = WindowFromPoint(pt);
        qr_DQRCtrlLocal.DropOn(qr_pDragWnd, pDropWnd);
        qr_DQRCtrlRemote.DropOn(qr_pDragWnd, pDropWnd);
        qr_pDragImage->DeleteImageList();
        delete qr_pDragImage;
        qr_pDragImage = NULL;
        qr_pDragWnd = NULL;
    }
}

```

```

        CDialog::OnLButtonUp(nFlags, point);
    }
void QueryRetrieve::OnMouseMove(UINT nFlags, CPoint point)
{
    if (qr_pDragImage && qr_pDragWnd) // In Drag&Drop mode ?
    {
        CPoint ptDropPoint(point);
        ClientToScreen(&ptDropPoint);
        qr_pDragImage->DragMove(ptDropPoint);
    }
    CDialog::OnMouseMove(nFlags, point);
}

/*****
*
*   Displaying remote task queue and scheduling dialogs
*
*****/
void QueryRetrieve::OnServicesShowRemoteTasks()
{
    qr_DQRCtrlRemote.ShowTaskView();
    UpdateWindow(); // Make sure it redraws the window
}

void QueryRetrieve::OnUpdateServicesShowRemoteTasks(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(qr_DQRCtrlRemote.HasTaskQueue());
}

void QueryRetrieve::OnServicesTaskScheduling()
{
    // Switch scheduling prompt
    qr_DQRCtrlRemote.SetTaskSchedulerPrompt(
        !qr_DQRCtrlRemote.GetTaskSchedulerPrompt());
    qr_UpdateMenu = true;
}

void QueryRetrieve::OnUpdateServicesTaskScheduling(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(qr_DQRCtrlRemote.GetTaskSchedulerPrompt());
}

```

```

// Server.h: interface for the Server class.
//
/////////////////////////////////////////////////////////////////

#ifndef _SERVER_H_INCLUDED_
#define _SERVER_H_INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class Server
{
public:
    bool RunServer(ApplicationEntity* a, bool multithread);

    Server(DICOMDatabase* db, DICOMViewLog* log);
    virtual ~Server();

private:
    UINT16      srv_CancelledID;
    DICOMViewLog* srv_pLog;
    DICOMDatabase* srv_pDataBase;
    struct Thread
    {
        char      tr_LocAET[20], tr_RemAET[20];
        UINT16*    tr_ptrCancelledID;
        int        tr_socket, tr_ServPort;
        static int tr_Count;
        DICOMViewLog* tr_pLog;
        DICOMDatabase* tr_pDataBase;

        void      StartThread(char* locAET, char* remAET, bool multithread);
        static void StartFunc(void *pThread);
        bool      RunThread();
        Thread (DICOMDatabase* db, DICOMViewLog* log, int sockfd);
        ~Thread() { tr_Count--; };
    };

    bool RunServer(char* port, char* locAET, char* remAET, bool multithread);
};

#endif // !defined(_SERVER_H_INCLUDED_)

```

```

// Server.cpp: implementation of the Server class.
//
/////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Server.h"
#include <process.h>

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif

/////////////////////////////////////////////////////////////////
// Server Class
/////////////////////////////////////////////////////////////////

Server::Server(DICOMDatabase* db, DICOMViewLog* log)
{
    srv_pDataBase=db;    srv_pLog=log;    srv_CancelledID=0;
}

Server::~Server()    {}

Server::Thread::Thread(DICOMDatabase* db, DICOMViewLog* log, int socketfd)
    : tr_pDataBase(db), tr_pLog(log), tr_socket(socketfd)
{
    tr_Count++;
    ZeroMem(tr_LocAET,20);    ZeroMem(tr_RemAET,20);
    tr_ServPort=0;
}

Server::Thread::~Thread()    { tr_Count = 0; }

/*
*****
* Start Server, listening to a specific port
*****
*/
bool Server::RunServer(char* port, char* locAET, char* remAET, bool multithread)
{
    // Do we have valid class members ?
    if(!srv_pDataBase || !srv_pLog) return false;
    if(!port || !locAET || !remAET)
    {
        srv_pLog->Load("Server ERROR: NULL parameter");
        return false;
    }
    char sport[8];    sprintf(sport,"%s",port);

    // Can we listen to the given port ?
    char    info[64];
    Socket    MSocket;
    if(!MSocket.Listen(sport))
    {
        sprintf(info,"Server ERROR: cannot listen to port %s",sport);
        srv_pLog->Load(info);
        return false;
    }
    sprintf(info,"Server listens to port %s",sport);
    srv_pLog->Load(info);

    // Run single or multithread server
    while(MSocket.Accept())
    {
        // Set server thread parameters
        Thread* trd = new Thread(srv_pDataBase, srv_pLog, MSocket.Socketfd);
        if(!trd)
        {
            srv_pLog->Load("Server ERROR: Out of memory for new thread");
            return false;
        }
        trd->tr_ServPort = atoi(sport);
        trd->tr_ptrCancelledID = &srv_CancelledID;
    }
}

```



```

    // Start server thread. This function will delete the thread "trd"
    // after completion - never use "thr" any more !
    trd->StartThread(locAET, remAET, multithread);
    MSocket.Socketfd = 0;
    MSocket.Connected = FALSE;
}
sprintf(info,"Server failed to accept more data at port %s",sport);
srv_pLog->Load(info);
return false;    // should never return except on error
}

bool Server::RunServer(ApplicationEntity* a, bool multithread)
{
    if(!a) return false;
    return RunServer(a->GetPortServerString(),a->ae_partnerTitle,
        a->ae_Title, multithread);
}

/*****
*
*   Start a thread, if possible
*
*****/
void Server::Thread::StartThread(char* locAET, char* remAET, bool multithread)
{
    strcpy(tr_LocAET,locAET);
    strcpy(tr_RemAET,remAET);
    if(multithread)
    {
        if(_beginthread(StartFunc, 100000, this) == -1)
        {
            try
            {
                tr_pLog->Load("Server WARNING: cannot start a new thread");
                RunThread();    // just try to run
            }
            catch (...) {}
        }
    }
    else    RunThread();

/*****
*
*   Run a thread, wrapped in C syntax
*
*****/
void Server::Thread::StartFunc(void *pThread)
{
    if(!pThread) return; // cannot be NULL
    ((Thread*)pThread)->RunThread();
    _endthread();
}

/*****
*
*   Run and DELETES a thread
*
*****/
bool Server::Thread::RunThread()
{
    if(!tr_pDataBase || ! tr_pLog)
    {
        delete this;
        return false;
    }
    char info[64];
    PDU_Service PDU;
    DICOMCommandObject DCO;
    UID uid;
    SCPProvider SCP(*tr_pLog, *tr_pDataBase);

    // Configure PDU
    PDU.ClearAbstractSyntaxs();
    PDU.SetLocalAddress((BYTE*)tr_LocAET);

```

```

PDU.SetRemoteAddress((BYTE*)tr_RemAET);
uid.Set("1.2.840.10008.3.1.1.1");      PDU.SetApplicationContext(uid);
PDU.AddStandardStoreAbstractSyntaxes();

// NOTE: PDU.Multiplex(filedes). This call is saying, here's a just
// connected socket, now go through the regular PDU association, and
// return to me a connected DICOM link.

if(PDU.Multiplex(tr_socket))
{
    while(TRUE)
    {
        tr_pLog->ReportTime();
        DCO.Reset();
        if(!PDU.Read(&DCO))
        {
            PDU.Close();
            delete this;
            return false;
        }
        sprintf(info, "Server object received at port %d:", tr_ServPort);
        tr_pLog->Load(info);
        tr_pLog->Load(DCO);
        Beep(300,100);
        if(!SCP.IdentifyAndProcess(PDU, DCO, tr_ptrCancelledID))    break;
    }    // while
}    // if
else
{
    switch (((AAssociateRJ)PDU).Reason)
    {
    case 3:
        tr_pLog->Load("Server ERROR: Rejected remote AE address");
        break;
    case 7:
        tr_pLog->Load("Server ERROR: Rejected local AE address");
        break;
    case 2:
        tr_pLog->Load("Server ERROR: Rejected proposed Application Context");
        break;
    default:
        tr_pLog->Load("Server ERROR: Reason unknown");
    }
    PDU.Close();
    delete this;
    return false;
}
// We get here on success
PDU.Close();
delete this;
return true;
}

```

```

// AccurateTimer.h: interface and implementation of the AccurateTimer class.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "DCM.h"

#ifdef _DEBUG
#ifdef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif
#endif

class AccurateTimer
{
private :
    int Initialized;
    __int64 Frequency;
    __int64 BeginTime;
public :
    AccurateTimer()        // constructor
    {
        // get the frequency of the counter
        Initialized = QueryPerformanceFrequency( (LARGE_INTEGER *)&Frequency );
    }

    BOOL Begin()           // start timing
    {
        if( ! Initialized ) return 0; // error - couldn't get frequency
        // get the starting counter value
        return QueryPerformanceCounter( (LARGE_INTEGER *)&BeginTime );
    }

    double End()           // stop timing and get elapsed time in seconds
    {
        if( ! Initialized ) return 0.0; // error - couldn't get frequency
        // get the ending counter value
        __int64 endtime;
        QueryPerformanceCounter( (LARGE_INTEGER *)&endtime );
        // determine the elapsed counts
        __int64 elapsed = endtime - BeginTime;
        // convert counts to time in seconds and return it
        return (double)elapsed / (double)Frequency;
    }

    void EndReport()        // stop timing and get elapsed time in seconds
    {
        double t=End();
        CString time; time.Format("Time=%lf seconds",t);
        AfxMessageBox(time);
    }

    BOOL Available() // returns true if the perf counter is available
    { return Initialized; }

    __int64 GetFreq() // return perf counter frequency as large int
    { return Frequency; }
};

```



```

// Angle.cpp: implementation of the Angle class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "DCM.h"
#include "Angle.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

Angle::Angle()
{
    Clean();
}

Angle::~Angle()
{
}

/*****
 * Report current angle in the status bar
 *****/
CString Angle::toString()
{
    CString info;
    info.Format("%.2lf ",a_angle);
    return (info+a_scale);
}

/*****
 * Compute current angle
 *****/
void Angle::GetAngle()
{
    a_angle=0.0;
    double x=_hypot(a_p1.x-a_p2.x,a_p1.y-a_p2.y);    if(x<=0.1) return;
    double y=_hypot(a_p3.x-a_p2.x,a_p3.y-a_p2.y);    if(y<=0.1) return;
    double z=_hypot(a_p1.x-a_p3.x,a_p1.y-a_p3.y);    if(z<=0.1) return;
    double cosa=(x*x+y*y-z*z)/(2*x*y);
    if(cosa>0.999) a_angle=3.1415926;
    else if(cosa<-0.999) a_angle=3.1415926;
    else a_angle=acos(cosa);
    a_angle *= a_scale_coeff;
    return;
}

/*****
 *
 * Reset all Angle parameters
 *
 *****/
void Angle::Clean()
{
    a_p1=CPoint(20,30);
    a_p2=CPoint(50,50);
    a_p3=CPoint(20,70);
    a_scale=CString("degrees");
    a_scale_coeff=180/3.1415926;
    GetAngle();
}

```

```

    a_active=false;
    a_undo=false;
}

/*****
*
*   Draw the Angle
*
*****/
void Angle::Draw(CDC *pDC)
{
    int dmode=SetROP2(pDC->m_hDC, R2_NOT);
    CPen* old_pen = pDC->SelectObject(&theApp.app_Pen);

    // Draw Angle lines
    pDC->MoveTo(a_p1); pDC->LineTo(a_p2);
    pDC->MoveTo(a_p2); pDC->LineTo(a_p3);
    // Circle the Angle points
    pDC->Arc(CRect(a_p1.x-6,a_p1.y-6,a_p1.x+6,a_p1.y+6), a_p1, a_p1);
    pDC->Arc(CRect(a_p2.x-6,a_p2.y-6,a_p2.x+6,a_p2.y+6), a_p2, a_p2);
    pDC->Arc(CRect(a_p2.x-4,a_p2.y-4,a_p2.x+4,a_p2.y+4), a_p2, a_p2);
    pDC->Arc(CRect(a_p3.x-6,a_p3.y-6,a_p3.x+6,a_p3.y+6), a_p3, a_p3);

    SetROP2(pDC->m_hDC, dmode);
    pDC->SelectObject(old_pen);
}

/*****
*
*   Update the Angle
*
*****/
void Angle::Redraw(CDC *pDC, CPoint &p, bool initialize)
{
    if(!a_active) return;

    // Initialize angle points
    if(initialize)
    {
        if(a_undo) Draw(pDC);
        int px=(p.x<<1)-10; int py=(p.y<<1)-10;
        if(a_p1.x>px) a_p1.x=px;    if(a_p1.y>py) a_p1.y=py;
        if(a_p3.x>px) a_p3.x=px;    if(a_p3.y>py) a_p3.y=py;
        a_p2=p;
        Draw(pDC); // new Angle
        GetAngle();
        a_undo=true;
        return;
    }

    // Find which Angle vertex is closer
    double x1=_hypot(p.x-a_p1.x,p.y-a_p1.y);
    double x2=_hypot(p.x-a_p2.x,p.y-a_p2.y);
    double x3=_hypot(p.x-a_p3.x,p.y-a_p3.y);

    // Update the nearest vertex
    if(x1<=x2 && x1<=x3)
    {
        if(x2<=10 || x3<=10) return; // avoid degraded Angle
        if(a_undo) Draw(pDC);
        a_p1=p;
    }
    else if(x2<=x1 && x2<=x3)
    {
        if(x1<=10 || x3<=10) return; // avoid degraded Angle
        if(a_undo) Draw(pDC);
        a_p2=p;
    }
    else
    {
        if(x1<=10 || x2<=10) return; // avoid degraded Angle
    }
}

```

```

        if(a_undo) Draw(pDC);
        a_p3=p;
    }

    Draw(pDC); // new Angle
    GetAngle();
    a_undo=true;
}

```

```

/*****
*
*   Update Angle scale
*
*****/
void Angle::SetScale(int code)
{

```

```

    switch(code)
    {
    case 1:
        a_scale="degrees";
        a_scale_coeff=180/3.1415926;
        break;
    case 2:
        a_scale="radians";
        a_scale_coeff=1.0;
        break;
    case 3:
        a_scale="percent";
        a_scale_coeff=50.0/3.1415926;
        break;
    }
}

```

```

/*****
*
*   Update Angle pop-up menu
*
*****/

```

```

void Angle::UpdatePopupMenu(CMenu *pop)
{
    GetAngle();
    if(a_scale=="degrees") pop->CheckMenuItem(ID_ANGLE_DEGREES,MF_CHECKED );
    else if (a_scale=="radians") pop->CheckMenuItem(ID_ANGLE_RADIANS,MF_CHECKED );
    else pop->CheckMenuItem(ID_ANGLE_PERCENT,MF_CHECKED );
    pop->ModifyMenu(ID_ANGLE_VALUE,MF_BYCOMMAND,ID_ANGLE_VALUE,toString());
}

```

```

// CustomFileDialog.h : header file
//

/////////////////////////////////////////////////////////////////
// CCustomFileDialog dialog
#include "resource.h"

#define CUSTOM_FILEOPENORD          1538
#define CUSTOM_MULTIFILEOPENORD    1539

class CCustomFileDialog : public CFileDialog
{
    DECLARE_DYNAMIC(CCustomFileDialog)
public:
    TCHAR          m_szBigBuffer[1000];
    static CString szCustomDefFilter;
    static CString szCustomDefExt;
    static CString szCustomDefFileName;
    static CString szCustomTitle;
    CStringList    m_listDisplayNames;

    void           SetTitle(CString title);
    int            GetSelectedCount() { return m_listDisplayNames.GetCount(); }
    CString        GetSelectedAt(UINT index);
    CCustomFileDialog(BOOL bOpenFileDialog = TRUE, // TRUE for FileOpen, FALSE for FileSaveAs
        DWORD dwFlags = OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT | OFN_NODEREFERENCELINKS | OFN_EXPL
ORER |
        OFN_ENABLETEMPLATE | OFN_ALLOWMULTISELECT | OFN_FILEMUSTEXIST,
        LPCTSTR lpszFilter = CCustomFileDialog::szCustomDefFilter,
        LPCTSTR lpszDefExt = CCustomFileDialog::szCustomDefExt,
        LPCTSTR lpszFileName = CCustomFileDialog::szCustomDefFileName,
        CWnd* pParentWnd = NULL);

    Dialog Data
    //{AFX_DATA(CCustomFileDialog)
    enum { IDD = IDD_CUSTOM_FILE_DIALOG };
    BOOL      m_bMulti;
    BOOL      m_SelectSubdirectories;
    //}AFX_DATA

    Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CCustomFileDialog)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}AFX_VIRTUAL
    virtual BOOL OnFileNameOK();

    // Implementation
protected:
    BOOL ReadListViewNames();    // protected -> not callable without dialog up

    //{AFX_MSG(CCustomFileDialog)
    afx_msg void OnSelectButton();
    afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
    virtual BOOL OnInitDialog();
    //}AFX_MSG
    afx_msg void OnHelp();
    DECLARE_MESSAGE_MAP()
};

```



```

switch(m_FromIndex)
{
case 0: // now
    m_StartTime = CTime::GetCurrentTime();
    break;
case 1: // in 10 min
    m_StartTime = CTime::GetCurrentTime() + CTimeSpan(0,0,10,0);
    break;
case 2: // in 30 min
    m_StartTime = CTime::GetCurrentTime() + CTimeSpan(0,0,30,0);
    break;
case 3: // in 1 hour
    m_StartTime = CTime::GetCurrentTime() + CTimeSpan(0,1,0,0);
    break;
case 4: // in 2 hours
    m_StartTime = CTime::GetCurrentTime() + CTimeSpan(0,2,0,0);
    break;
case 5: // tomorrow 8:00 am
    tm = CTime::GetCurrentTime() + CTimeSpan(1,0,0,0);
    m_StartTime = CTime(tm.GetYear(), tm.GetMonth(), tm.GetDay(),
                        8,0,0);
    break;
case 6: // specific
    tm = m_StartTime;
    m_StartTime = CTime(m_StartDate.GetYear(), m_StartDate.GetMonth(),
                        m_StartDate.GetDay(), tm.GetHour(),
                        tm.GetMinute(), tm.GetSecond());
    if(m_StartTime < CTime::GetCurrentTime())
    {
        m_StartTime = CTime::GetCurrentTime();
    }
    break;
}
// 2. Find out end date/time
switch(m_ToIndex)
{
case 0: // undefined
    m_EndTime = CTime(2030,1,1,1,1,1);
    break;
case 1: // in 10 min
    m_EndTime = m_StartTime + CTimeSpan(0,0,10,0);
    break;
case 2: // in 30 min
    m_EndTime = m_StartTime + CTimeSpan(0,0,30,0);
    break;
case 3: // in 1 hour
    m_EndTime = m_StartTime + CTimeSpan(0,1,0,0);
    break;
case 4: // in 2 hours
    m_EndTime = m_StartTime + CTimeSpan(0,2,0,0);
    break;
case 5: // tomorrow 8:00 am
    tm = CTime::GetCurrentTime() + CTimeSpan(1,0,0,0);
    m_EndTime = CTime(tm.GetYear(), tm.GetMonth(), tm.GetDay(),
                        8,0,0);
    if(m_EndTime <= m_StartTime )
    {
        m_EndTime = m_StartTime + CTimeSpan(0,0,30,0);
    }
    break;
case 6: // specific
    tm = m_EndTime;
    m_EndTime = CTime(m_EndDate.GetYear(), m_EndDate.GetMonth(),
                        m_EndDate.GetDay(), tm.GetHour(),
                        tm.GetMinute(), tm.GetSecond());
    if(m_EndTime <= m_StartTime )
    {
        m_EndTime = m_StartTime + CTimeSpan(0,0,30,0);
    }
    break;
}

// 3. Set task schedule
DateTime      dstart, dend;

```

```

dstart.SetDateTime(m_StartTime.GetYear(), m_StartTime.GetMonth()-1,
    m_StartTime.GetDay()-1, m_StartTime.GetHour(), m_StartTime.GetMinute(),
    m_StartTime.GetSecond());
if(m_FromIndex>0)
{
    dend.SetDateTime(m_EndTime.GetYear(), m_EndTime.GetMonth()-1,
        m_EndTime.GetDay()-1, m_EndTime.GetHour(), m_EndTime.GetMinute(),
        m_EndTime.GetSecond());
}
t.ScheduleTask(dstart,dend);

// 4. Set number of execution attempts
t.SetExec(m_Attempts);
}

```

```

/*****
*
*   Display schedule dialog and schedule the task
*
*****/
void DQRTaskSchedule::RunScheduler(DQRTask &t, bool prompt)
{
    if(prompt)
    {
        if(DoModal() != IDOK)    return;
    }
    ScheduleTask(t);
}

```

1

```

CString m_SeriesNum;
CString m_PatientID2;
CString m_StudyInstUID2;
CString m_StudyID;
CString m_AccessionNumber;
double m_StudyTime;
long m_StudyDate;
CString m_StudyImagesNum;
//}}AFX_FIELD

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(ODBC4Set)
public:
virtual CString GetDefaultSQL(); // Default SQL for Recordset
virtual void DoFieldExchange(CFieldExchange* pFX); // RFX support
//}}AFX_VIRTUAL

// Implementation
#ifdef _DEBUG
#endif
private:
void ClearSet();
void SetFindFilter(DICOMRecord& dr);
void UpdateFromDICOMRecord(DICOMRecord &dr) { ; } // not used
bool SetFromDICOMRecord(DICOMRecord &dr) { return true; } // not used
CString GetDefaultQuery(DICOMRecord& dr);

class ODBCDatabase : public DICOMDatabase
public:
static const CString db_DBName;

void RemoveAllRecords();
void StopSearch(void* pTR);
void DisplayRecords(Array<DICOMRecord> &a, bool from_local);
bool InitializeDataBase(char* directory,
void (*disp)(Array<DICOMRecord>&, bool));
bool DBAdd(DICOMRecord& dr);
bool DBAdd(DICOMObject* dob, PDU_Service* pdu);
bool GetFromLocal(DICOMDataObject& ddo_mask);
bool SetRecordCount(int c) { return true; }; // unused
BYTE MatchNext(void* pTR, DICOMObject &dob_found,
DICOMObject *dob_mask);
BYTE RetrieveNext(void* pTR, DICOMObject& dob_found);
int DBRemove(DICOMRecord& dr_mask);
int GetRecordCount() { return 1; }; // unused
CDatabase* GetCDatabasePtr() { return &db_ODBCdb; };
ODBCDatabase();
virtual ~ODBCDatabase();

protected:
void* StartSearch(DICOMRecord& dr_mask, const BYTE how);
bool DBAddDirectoryContents(char* directory, bool copy_files,
bool include_subdirectories=true);
int DBRemoveDirectoryContents(char *directory, bool use_filenames,
bool include_subdirectories=true);

private:
bool db_IsODBC;
CDatabase db_ODBCdb;
CCriticalSection
db_DBAddDDO_CriticalSection, db_DisplayRecords_CriticalSection;

bool RemoveUnique(CString& pKey, CString& stKey,
CString& serKey, CString& imKey);
bool Connect();
};

////////////////////////////////////
//

```

```

// ODBCPatientSet recordset
//
/////////////////////////////////////////////////////////////////
class ODBCPatientSet : public ODBCTableSet
{
public:
    void        WriteIntoDICOMRecord(DICOMRecord& dr);
    void        SetFindFilter(DICOMRecord& dr);
    ODBCPatientSet(CDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(ODBCPatientSet)

// Field/Param Data
//{{AFX_FIELD(ODBCPatientSet, CRecordset)
CString m_PatientID;
CString m_PatientName;
double m_PBirthTime;
long m_PBirthDate;
//}}AFX_FIELD

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(ODBCPatientSet)
public:
    virtual CString GetDefaultSQL();    // Default SQL for Recordset
    virtual void DoFieldExchange(CFieldExchange* pFX); // RFX support
//}}AFX_VIRTUAL

// Implementation
private:
    void        ClearSet();
    void        UpdateFromDICOMRecord(DICOMRecord& dr);
    bool        SetFromDICOMRecord(DICOMRecord &dr);
    CString     GetDefaultQuery(DICOMRecord& dr);

/////////////////////////////////////////////////////////////////
// ODBCStudySet recordset
//
/////////////////////////////////////////////////////////////////
class ODBCStudySet : public ODBCTableSet
{
public:
    void        WriteIntoDICOMRecord(DICOMRecord& dr);
    void        SetFindFilter(DICOMRecord &dr);
    ODBCStudySet(CDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(ODBCStudySet)

// Field/Param Data
//{{AFX_FIELD(ODBCStudySet, CRecordset)
CString m_PatientID;
CString m_StudyInstUID;
CString m_StudyID;
CString m_AccessionNumber;
double m_StudyTime;
long m_StudyDate;
CString m_StudyImagesNum;
//}}AFX_FIELD

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(ODBCStudySet)
public:
    virtual CString GetDefaultSQL();    // Default SQL for Recordset
    virtual void DoFieldExchange(CFieldExchange* pFX); // RFX support
//}}AFX_VIRTUAL
private:
    void        UpdateFromDICOMRecord(DICOMRecord &dr);
    void        ClearSet();
    bool        SetFromDICOMRecord(DICOMRecord &dr);
    CString     GetDefaultQuery(DICOMRecord& dr);

```

```

};

/////////////////////////////////////////////////////////////////
//
// ODBCSeriesSet recordset
//
/////////////////////////////////////////////////////////////////
class ODBCSeriesSet : public ODBCTableSet
{
public:
    void        WriteIntoDICOMRecord(DICOMRecord &dr);
    void        SetFindFilter(DICOMRecord &dr);
    ODBCSeriesSet(CDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(ODBCSeriesSet)

// Field/Param Data
//{{AFX_FIELD(ODBCSeriesSet, CRecordset)
CString m_StudyInstUID;
CString m_SeriesInstUID;
CString m_Modality;
CString m_SeriesNum;
//}}AFX_FIELD

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(ODBCSeriesSet)
public:
    virtual CString GetDefaultSQL();    // Default SQL for Recordset
    virtual void DoFieldExchange(CFieldExchange* pFX);  // RFX support
//}}AFX_VIRTUAL
private:
    void        UpdateFromDICOMRecord(DICOMRecord &dr);
    void        ClearSet();
    bool        SetFromDICOMRecord(DICOMRecord &dr);
    CString     GetDefaultQuery(DICOMRecord &dr);
};

/////////////////////////////////////////////////////////////////
//
// ODBCImageSet recordset
//
/////////////////////////////////////////////////////////////////
class ODBCImageSet : public ODBCTableSet
{
public:
    void        WriteIntoDICOMRecord(DICOMRecord &dr);
    void        SetFindFilter(DICOMRecord &dr);
    CString     GetFilename() { return m_Filename; };
    ODBCImageSet(CDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(ODBCImageSet)

// Field/Param Data
//{{AFX_FIELD(ODBCImageSet, CRecordset)
CString m_SeriesInstUID;
CString m_SOPInstUID;
CString m_ImageNum;
CString m_Filename;
//}}AFX_FIELD

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(ODBCImageSet)
public:
    virtual CString GetDefaultSQL();    // Default SQL for Recordset
    virtual void DoFieldExchange(CFieldExchange* pFX);  // RFX support
//}}AFX_VIRTUAL
private:
    void        UpdateFromDICOMRecord(DICOMRecord &dr);
    void        ClearSet();
    bool        SetFromDICOMRecord(DICOMRecord &dr);
    CString     GetDefaultQuery(DICOMRecord &dr);
};

```

```
#endif // !defined(AFX_ODBC_H_INCLUDED_)
```

```

////////////////////////////////////
// ODBCDatabase Class
// This class is derived from DICOMDatabase to support
// ODBC data sources
////////////////////////////////////
#include "stdafx.h"
#include <odbcinst.h>
#include "ODBC.h"
#include "../DCM.h"

////////////////////////////////////
// Construction/Destruction
////////////////////////////////////
const CString ODBCDatabase::db_DBName = "DCMDBase";

ODBCDatabase::ODBCDatabase()
{
    db_IsODBC=false;
}

ODBCDatabase::~ODBCDatabase()
{
    // Close database connection
    TRY { if(db_ODBCdb.IsOpen()) db_ODBCdb.Close(); }
    CATCH(CException, e) { ; }
    END_CATCH;
}

/*****
*
* Add records
*****/
bool ODBCDatabase::DBAdd(DICOMRecord &dr)
{
    if(!db_IsODBC) return DICOMDatabase::DBAdd(dr);
    if(!dr.HasUniquePrimaryKeys()) return false;
    // Check for consistency
    ODBC4Set set;
    if(set.HasAliasRecords(dr)) return false; // violates DB tree structure
    // Insert
    ODBCPatientSet ps;
    if(!ps.AddOrUpdate(dr)) return false;
    ODBCStudySet sts;
    if(!sts.AddOrUpdate(dr)) return false;
    OBCSeriesSet srs;
    if(!srs.AddOrUpdate(dr)) return false;
    ODBCImageSet ims;
    if(!ims.AddOrUpdate(dr)) return false;
    db_MostRecentRecord=dr;
    return true;
}

/*****
*
* Add objects, thread-safe
*****/
bool ODBCDatabase::DBAdd(DICOMObject *dob, PDU_Service *pdu)
{
    bool success = false;
    CSingleLock singleLock(&db_DBAddDDO_CriticalSection);
    singleLock.Lock(3000); // attempt to lock the shared resource
    if (singleLock.IsLocked()) // resource has been locked
    {
        success = DICOMDatabase::DBAdd(dob, pdu);
    }
    singleLock.Unlock();
    return success;
}

/*****
*
* Display records, thread-safe
*****/

```



```

*****/
void ODBCDatabase::DisplayRecords(Array<DICOMRecord> &a, bool from_local)
{
    CSingleLock singleLock(&db_DisplayRecords_CriticalSection);
    singleLock.Lock(10000); // attempt to lock the shared resource
    if (singleLock.IsLocked()) // resource has been locked
    {
        DICOMDatabase::DisplayRecords(a, from_local);
    }
    singleLock.Unlock();
}
/*****
*
*   Add files and directories
*
*****/
bool ODBCDatabase::DBAddDirectoryContents(char *directory, bool copy_files,
                                          bool include_subdirectories/*=true*/)
{
    CString info("Adding directory ");
    info += CString(directory);

    // Process individual file
    if(DICOMDatabase::DBAddDirectoryContents(directory, copy_files,
        include_subdirectories)) return true;

    // Processing Windows directory
    WIN32_FIND_DATA wf;
    // Add '\\\' to the end of directory string, if needed
    char dir[MAX_PATH];
    char clast = directory[strlen(directory)-1];
    if(clast=='/' || clast=='\\') sprintf(dir,"%s",directory);
    else sprintf(dir,"%s\\",directory);

    // Set wildcard search mask
    char nf[MAX_PATH];
    sprintf(nf,"%s*",dir);

    // Search
    char fullname[MAX_PATH];
    HANDLE hf=FindFirstFile(nf,&wf);
    if(hf==INVALID_HANDLE_VALUE) return false;
    do
    {
        if(strcmp(wf.cFileName,".")==0) continue;
        if(strcmp(wf.cFileName,"..")==0) continue;
        sprintf(fullname,"%s%s",dir,wf.cFileName);
        if(wf.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
        {
            if(include_subdirectories)
            {
                DBAddDirectoryContents(fullname, copy_files);
            }
        }
        else
        {
            DICOMDatabase::DBAdd(fullname, copy_files);
        }
    }
    while (FindNextFile(hf,&wf));
    FindClose(hf);
    return true;
}
/*****
*
*   Remove files and directories
*
*****/
int ODBCDatabase::DBRemoveDirectoryContents(char *directory,
                                          bool use_filenames, bool include_subdirectories/*=true*/)
{
    CString info("Removing directory ");
    info += CString(directory);

    // Process individual file
    if(DICOMDatabase::DBRemoveDirectoryContents(directory, use_filenames,
        include_subdirectories)) return true;
}

```

```

// Processing Windows directory
WIN32_FIND_DATA wf;
// Add '\\\' to the end of directory string, if needed
char dir[MAX_PATH];
char clast = directory[strlen(directory)-1];
if(clast=='/' || clast=='\\')    sprintf(dir,"%s",directory);
else                            sprintf(dir,"%s\\",directory);
// Set wildcard search mask
char nf[MAX_PATH];
sprintf(nf,"%s*",dir);
// Search
char fullname[MAX_PATH];
HANDLE hf=FindFirstFile(nf,&wf);
if(hf==INVALID_HANDLE_VALUE)    return false;
do
{
    if(strcmp(wf.cFileName,".")==0)    continue;
    if(strcmp(wf.cFileName,"..")==0)    continue;
    sprintf(fullname,"%s%s",dir,wf.cFileName);
    if(wf.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
    {
        if(include_subdirectories)
        {
            DBRemoveDirectoryContents(fullname, use_filenames);
        }
    }
    else
    {
        DICOMDatabase::DBRemove(fullname, use_filenames);
    }
}
while (FindNextFile(hf,&wf));
FindClose(hf);
return true;
}

*****
Remove all records from the database
*****
void ODBCDatabase::RemoveAllRecords()
{
    if(AfxMessageBox("Are you sure you want to remove all database records ?",
        MB_YESNO) == IDNO)    return;
    DICOMRecord dr;
    int n = DBRemove(dr);
    if(n>0)
    {
        CString info;
        info.Format("%d records were removed", n);
        AfxMessageBox(info, MB_ICONINFORMATION);
    }
    else AfxMessageBox("No records found", MB_ICONINFORMATION);
}

/*****
*
*   Open local records
*
*****/
bool ODBCDatabase::GetFromLocal(DICOMDataObject &dco_mask)
{
    if(!db_IsODBC)    return DICOMDatabase::GetFromLocal(dco_mask);
    ODBCTableSet* pTR=NULL;
    pTR = (ODBCTableSet*)DICOMDatabase::StartSearch(dco_mask,
        RetrieveHierarchical);
    if(!pTR)    return false;
    Array<DICOMRecord>    found;
    CString    fname;
    do
    {
        fname = pTR->GetFilename();
        if(fname=="")    continue;

```

```

    pTR->WriteIntoDICOMRecord(db_MostRecentRecord);
    found.Add(db_MostRecentRecord);
}
while(pTR->FetchNextRecord());
StopSearch(pTR);
DisplayRecords(found, true);
return true;
}

/*****
*
*   Initialize database
*
*****/
bool ODBCDatabase::InitializeDataBase(char *directory,
                                     void (__cdecl *disp)(Array<DICOMRecord>&, bool))
{
    if(!DICOMDatabase::InitializeDataBase(directory, disp)) return false;
    // Try to open ODBC connection
    if(Connect()) return true;
    // Try to find the local DB file
    CString failedODBC = CString("DCM failed to setup ODBC database.\n")+
        CString("File-based database will be used instead");
    CString dbfilenew = CString(directory);
    int n = dbfilenew.Find("\\\\Applic",0); if(n>0) dbfilenew=dbfilenew.Left(n);
    CString dbfileold;
    dbfileold.Format("%s\\%s.mdb",dbfilenew,db_DBName);
    dbfilenew.Format("%s\\%s.mdb",directory,db_DBName);
    if(GetFileAttributes(dbfilenew)==-1) // no such file
    {
        if(CopyFile(dbfileold, dbfilenew, FALSE) == FALSE)
        {
            CString nofile;
            nofile.Format("Cannot find ODBC database file %s.\n",dbfileold);
            AfxMessageBox(nofile+failedODBC, MB_ICONINFORMATION|MB_OK);
            return true;
        }
    }
    // Try to reinstall the ODBC
    CString driver = "Microsoft Access Driver (*.mdb)";
    CString config;
    config.Format("DSN=%s; DESCRIPTION=DCM Application Database;"
        "DBQ=%s;",db_DBName, dbfilenew);

    if(::SQLConfigDataSource(NULL,ODBC_ADD_DSN,
        driver,config) == FALSE)
    {
        AfxMessageBox(failedODBC, MB_ICONINFORMATION|MB_OK);
    }
    else
    {
        if(!Connect()) AfxMessageBox(failedODBC, MB_ICONINFORMATION|MB_OK);
    }
    return true;
}

/*****
*
*   Remove records. Returns the number of deleted records
*
*****/
int ODBCDatabase::DBRemove(DICOMRecord &dr_mask)
{
    if(!db_IsODBC) return DICOMDatabase::DBRemove(dr_mask);
    ODBC4Set* pTR=NULL;
    pTR = (ODBC4Set*)StartSearch(dr_mask, RetrieveRelational);
    if(!pTR) return false;
    int removed=0;
    do
    {
        RemoveUnique(pTR->m_PatientID, pTR->m_StudyInstUID,
            pTR->m_SeriesInstUID, pTR->m_SOPInstUID);
        removed++;
    }
}

```

```

while(pTR->FetchNextRecord());
StopSearch(pTR);
return removed;
}

/*****
*
*   Search records
*
*****/
void* ODBCDatabase::StartSearch(DICOMRecord &dr_mask, const BYTE how)
{
    if(!db_IsODBC) return DICOMDatabase::StartSearch(dr_mask, how);
    // Initialize appropriate class pointer for match/retrieve
    ODBCTableSet* pTR = NULL;
    TRY
    {
        if(how==MatchRelational || how==RetrieveRelational)
        {
            ODBC4Set* pSet = new ODBC4Set();
            pTR = pSet;
        }
        else if(how==MatchHierarchical)
        {
            BYTE lev = dr_mask.FindQLevel();
            if(lev==DICOMRecord::LevelInvalid)
            {
                ODBCPatientSet* pSet = new ODBCPatientSet();
                pTR = pSet;
            }
            else if(lev==DICOMRecord::LevelPatient)
            {
                ODBCStudySet* pSet = new ODBCStudySet();
                pTR = pSet;
            }
            else if(lev==DICOMRecord::LevelStudy)
            {
                OBCSeriesSet* pSet = new OBCSeriesSet();
                pTR = pSet;
            }
            else // lev==LevelSeries or lev==LevelImage
            {
                ODBCImageSet* pSet = new ODBCImageSet();
                pTR = pSet;
            }
        }
        else if(how==RetrieveHierarchical)
        {
            if(!::IsUniqueString(dr_mask.GetSOPInstUID()))
            {
                return ODBCDatabase::StartSearch(dr_mask, RetrieveRelational);
            }
            ODBCImageSet* pSet = new ODBCImageSet();
            pTR = pSet;
        }
        else return NULL;
        if(!pTR) return NULL;
        pTR->SetFindFilter(dr_mask);
        if(!pTR->Open()) { delete pTR; return NULL; }
        if(pTR->IsEOF()) { pTR->Close(); delete pTR; return NULL; }
        return pTR;
    }
    // TRY
    CATCH(CException, e)
    {
        if(pTR) delete pTR;
        return NULL;
    }
    END_CATCH;
    if(pTR) delete pTR;
    return NULL;
}

void ODBCDatabase::StopSearch(void* pTR)
{
    if(!db_IsODBC) DICOMDatabase::StopSearch(pTR);
}

```

```

    try { delete (ODBCTableSet*) pTR; }
    catch(...) { }
}

BYTE ODBCDatabase::RetrieveNext(void* pTR, DICOMObject &dob_found)
{
    if(!db_IsODBC) return DICOMDatabase::RetrieveNext(pTR, dob_found);
    if(!pTR) return RecordsEnd;
    ODBCTableSet* pTabRec = (ODBCTableSet*)pTR;
    if(!pTabRec->IsOpen() || pTabRec->IsEOF()) return RecordsEnd;
    bool loaded = dob_found.LoadFromFile((char*)(LPCSTR)pTabRec->GetFilename());
    pTabRec->FetchNextRecord();
    if(loaded) return RecordFound;
    else return FindMore;
}

BYTE ODBCDatabase::MatchNext(void* pTR, DICOMObject &dob_found,
                             DICOMObject *dob_mask)
{
    if(!db_IsODBC) return DICOMDatabase::MatchNext(pTR, dob_found, dob_mask);
    if(!pTR) return RecordsEnd;
    ODBCTableSet* pTabRec = (ODBCTableSet*)pTR;
    if(!pTabRec->IsOpen() || pTabRec->IsEOF()) return RecordsEnd;
    pTabRec->WriteIntoDICOMObject(dob_found, dob_mask);
    pTabRec->FetchNextRecord();
    return RecordFound;
}

/*****
 *
 * Connect to the application database
 *
 *****/
bool ODBCDatabase::Connect()
{
    db_IsODBC=false;
    TRY
    {
        if(db_ODBCdb.IsOpen()) db_ODBCdb.Close();
        CString con;
        con.Format("DSN=%s;", db_DBName);
        db_IsODBC = (db_ODBCdb.OpenEx(con, CDatabase::noOdbcDialog)==TRUE);
        return db_IsODBC;
    }
    CATCH(CException, e)
    {
        #ifdef _DEBUG
        e->ReportError();
        #endif
        return db_IsODBC;
    }
    END_CATCH;
    return db_IsODBC;
}

/*****
 *
 * Remove an entry with unique primary keys
 *
 *****/
bool ODBCDatabase::RemoveUnique(CString &pKey, CString &stKey,
                                CString &serKey, CString &imKey)
{
    bool stop;
    TRY
    {
        // Remove at image level
        ODBCImageSet ims;
        ims.m_strFilter.Format("SeriesInstUID='%s'", serKey);
        if(!ims.Open()) return false;
        if(!ims.CanUpdate()) { ims.Close(); return false; }
        stop = false;
        while(!ims.IsEOF())
        {
            if(ims.m_SOPInstUID!=imKey) stop=true;
            else
            {
                if(ims.m_Filename.Find(db_Directory,0)>=0)

```

```

        {
            DeleteFile(ims.m_FileName);
        }
        ims.Delete();
    }
    ims.MoveNext();
}
ims.Close();
if(stop)    return true;

// Remove at series level
ODBCSeriesSet srs;
srs.m_strFilter.Format("StudyInstUID='%s'", stKey);
if(!srs.Open()) return false;
if(!srs.CanUpdate( )) {    srs.Close();    return false;    }
stop = false;
while(!srs.IsEOF())
{
    if(srs.m_SeriesInstUID != serKey)    stop=true;
    else    srs.Delete();
    srs.MoveNext();
}
srs.Close();
if(stop)    return true;

// Remove at study level
ODBCStudySet sts;
sts.m_strFilter.Format("PatientID='%s'", pKey);
if(!sts.Open()) return false;
if(!sts.CanUpdate( )) {    sts.Close();    return false;    }
stop = false;
while(!sts.IsEOF())
{
    if(sts.m_StudyInstUID != stKey) stop=true;
    else    sts.Delete();
    sts.MoveNext();
}
sts.Close();
if(stop)    return true;

// Remove at patient level
ODBCPatientSet ps;
ps.m_strFilter.Format("PatientID='%s'", pKey);
if(!ps.Open())    return false;
if(!ps.CanUpdate( ))    {    ps.Close(); return false;    }
while(!ps.IsEOF())
{
    ps.Delete();
    if(!ps.IsEOF()) ps.MoveNext();
}
ps.Close();

return true;
}
CATCH(CException, e)
{
    #ifdef _DEBUG
    e->ReportError();
    #endif
    return false;
}
END_CATCH;
return false;
}
////////////////////////////////////
//
// ODBCTableSet
//
////////////////////////////////////

IMPLEMENT_DYNAMIC(ODBCTableSet, CRecordset)

ODBCTableSet::ODBCTableSet(CDatabase* pdb)
: CRecordset(theApp.app_DataBase.GetCDatabasePtr())

```

```

{
    m_nDefaultType = dynaset;
}
// Smart record-closing destructor
ODBCTableSet::~ODBCTableSet()
{
    TRY { if(IsOpen()) Close(); }
    CATCH(CException, e) { ; }
    END_CATCH;
}

CString ODBCTableSet::GetDefaultConnect()
{
    CString connect;
    connect.Format("ODBC;DSN=%s", theApp.app_DataBase.db_DBName);
    return connect;
}

#ifdef _DEBUG
void ODBCTableSet::AssertValid() const
{
    CRecordset::AssertValid();
}

void ODBCTableSet::Dump(CDumpContext& dc) const
{
    CRecordset::Dump(dc);
}

#endif // _DEBUG
/* *****
 *
 * Moving to next record, if possible
 * *****/
bool ODBCTableSet::FetchNextRecord()
{
    TRY
    {
        if(!IsOpen()) return false;
        MoveNext();
        if(IsEOF()) { Close(); return false; }
        return true;
    }
    CATCH(CException, e) { return false; }
    END_CATCH;
    return false;
}
/* *****
 *
 * Append filter criteria
 * *****/
void ODBCTableSet::AndFilter(CString fName, char *sValue)
{
    if(::IsEmptyString(sValue)) return;
    CString fValue = ::Trim(sValue);
    if(fValue=="") return;

    CString filter("");
    CString and = (m_strFilter == "" ? "" : " AND");
    if(fName.Find("Filename",0)<0 && fValue.Find('\\',0)>0) // we have a set
    {
        fValue.Replace("\\", "\\");
        filter.Format("%s %s IN ('%s') ", and, fName, fValue);
        filter.Replace('*', '%'); filter.Replace('?', '_');
    }
    else if(fValue.Find('*',0)>=0 || fValue.Find('?',0)>=0) // we have wildcards
    {
        if(fValue=="*") return; // anything goes
        filter.Format("%s %s LIKE '%s' ", and, fName, fValue);
        filter.Replace('*', '%'); filter.Replace('?', '_');
    }
    else // plain match
    {

```

```

        Beep(500,100);
        if(!GetSafeHwnd()) return;
        if(m_SearchDialog.DoModal() != IDOK) return;
        Beep(500,100);
        m_RequestedClientService=find_root;
        if(!AfxBeginThread(StartQRClient,this,
            GetThreadPriority(),m_ClientStackSize)) RunClientThread();
    }
}

void DQRControl::FindAll()
{
    if(!m_LocalOnly) m_PreloadData=false; // we preload only on local !
    if(m_PreloadData)
    {
        m_RequestedClientService=find_root;
        if(!AfxBeginThread(StartQRClient,this,GetThreadPriority(),
            m_ClientStackSize)) RunClientThread();
    }
}

void DQRControl::UpdateAfterFind(bool find_success)
{
    LoadFoundList(!find_success);
    if(!find_success) AfxMessageBox("Cannot find", MB_ICONEXCLAMATION|MB_OK);
    if(find_success && m_DQR.GetFoundCount()>0) // Something's found
    {
        // Set list selection
        if(m_RequestedClientService=find_previous)
        {
            int nsel=m_ResultsListSelection[max(1,m_DQR.GetLevel())-1];
            if(nsel<0 || nsel>m_DQR.GetFoundCount())
            {
                nsel=0;
                m_ButtonGet.EnableWindow(FALSE);
                m_ButtonMoveTo.EnableWindow(FALSE);
            }
            else
            {
                m_ButtonGet.EnableWindow(nsel>0);
                m_ButtonMoveTo.EnableWindow(nsel>0);
            }
            m_ResultsList.SetItemState(nsel,LVIS_SELECTED | LVIS_FOCUSED , LVIS_SELECTED | LVIS_FOCUSED);
            m_ResultsList.EnsureVisible(nsel,FALSE);
        }
    }
}

*****
C-Get
*****

void DQRControl::OnButtonGet()
{
    Beep(500,100);
    UpdateData(TRUE);
    int sel;
    if(GetResultsListSelection(sel, m_DOBIndex)<2)
    {
        AfxMessageBox("Cannot get unspecified entry");
        return;
    }
    m_RequestedClientService=get_index;
    if(m_UseTaskQueue)
    {
        m_DQR.Get(m_DOBIndex, true); // queue
    }
    else
    {
        if(!AfxBeginThread(StartQRClient,this,GetThreadPriority(),
            m_ClientStackSize)) RunClientThread();
    }
}

```



```

/*****
*
*           C-Move
*
*****/
void DQRControl::OnButtonMoveTo()
{
    Beep(500,100);
    UpdateData(TRUE);
    int sel;
    if(GetResultsListSelection(sel, m_DOBindex)<2)
    {
        AfxMessageBox("Cannot move unspecified entry");
        return;
    }
    UINT n=m_MoveArchiveList.GetCurSel();
    if(!m_AEarray ) return;
    strcpy(m_MoveDestinationAE, m_AEarray->Get(n).ae_Title);
    m_RequestedClientService=move_index;
    if(m_UseTaskQueue)
    {
        m_DQR.Move(m_DOBindex,m_MoveDestinationAE,true);    // queue
    }
    else
    {
        if(!AfxBeginThread(StartQRClient,this,GetThreadPriority(),
            m_ClientStackSize)) RunClientThread();
    }
}

/*****
*
* Run a CLIENT thread, wrapped in C syntax
*
*****/
UINT DQRControl::StartQRClient(LPVOID ptrDQRControl)
{
    DQRControl* pDQRC=(DQRControl*)ptrDQRControl;
    if(pDQRC==NULL) return (UINT)FALSE; // illegal parameter
    return pDQRC->RunClientThread();
}

/*****
*
* Thread-handling routines
*
*****/
void DQRControl::SetInterruptMode(bool interrupt)
{
    int sw = interrupt ? SW_HIDE : SW_SHOW;
    GetDlgItem(IDC_STATIC_MOVE_TO)->ShowWindow(sw);
    if(m_LocalOnly)
    {
        GetDlgItem(IDC_BUTTON_DELETE)->ShowWindow(sw);
    }

    m_ArchiveList.EnableWindow(!interrupt && !m_LocalOnly);
    m_MoveArchiveList.ShowWindow(sw);
    m_ResultsList.EnableWindow(!interrupt);

    m_ButtonSearch.ShowWindow(sw);
    m_ButtonGet.ShowWindow(sw);
    m_ButtonMoveTo.ShowWindow(sw);

    // Enable animation
    if(interrupt)
    {
        m_AnimNetwork.ShowWindow(SW_SHOW);
        GetDlgItem(IDC_STATIC_PROGRESS)->ShowWindow(SW_SHOW);
        m_AnimNetwork.Open(IDR_AVI_CONNECT);
    }
    else

```

```

{
    m_AnimNetwork.Close();
    m_AnimNetwork.ShowWindow(SW_HIDE);
    m_ButtonCancel.ShowWindow(SW_HIDE);
    GetDlgItem(IDC_STATIC_PROGRESS)->SetWindowText("");
    GetDlgItem(IDC_STATIC_PROGRESS)->ShowWindow(SW_HIDE);
}

/*****
*
*   Thread-safe UpdateData
*
*****/
BOOL DQRControl::UpdateData(BOOL bSaveAndValidate)
{
    if(m_HWND) return FromHandle(m_HWND)->UpdateData(bSaveAndValidate);
    else return FALSE;
}

/*****
*
*   Run client thread
*
*****/
bool DQRControl::RunClientThread()
{
    //Disable window controls
    SetInterruptMode(true);
    // Lock on thread start
    CSingleLock singleLock(&m_RunClientThread_CritSection);
    singleLock.Lock(3000); // attempt to lock the shared resource

    m_DQR.SetLastMessageID(0);

    // Execute DIMSE service
    bool success=false;
    switch(m_RequestedClientService)
    {
    case echo: // C-Echo
        GetDlgItem(IDC_CONNECTION)->
            SetWindowText("Archive connection: Verifying ...");
        success = m_DQR.Echo();
        if(success) GetDlgItem(IDC_CONNECTION)->
            SetWindowText("Archive connection: Connected");
        else
            GetDlgItem(IDC_CONNECTION)->
                SetWindowText("Archive connection: Cannot connect");
        break;
    case find_root:
        {
            DICOMRecord dr;
            m_SearchDialog.WriteIntoDICOMRecord(dr);
            success=m_DQR.FindRoot(dr);
        }
        UpdateAfterFind(success);
        break;
    case find_previous:
        success=m_DQR.FindPreviousLevel();
        UpdateAfterFind(success);
        break;
    case find_next:
        success=m_DQR.FindNextLevel(m_DOBIndex);
        UpdateAfterFind(success);
        break;
    case find:
        if(m_DQR.Find()) UpdateAfterFind(true);
        else
        {
            CString info;
            info.Format("Several records were removed from the local database\n\n"
                "Some of the records in the query results window\n"
                "may no longer be valid !");
            AfxMessageBox(info, MB_ICONINFORMATION);
        }
        break;
    case get_index:

```

```

        success=m_DQR.Get(m_DOBindex, false);    // no queue
        LoadFoundList(false);
        if(!success)      AfxMessageBox("Cannot get", MB_ICONEXCLAMATION|MB_OK);
        break;
    case move_index:
        success=m_DQR.Move(m_DOBindex,m_MoveDestinationAE,false);    // no queue
        LoadFoundList(false);
        if(!success)      AfxMessageBox("Cannot move", MB_ICONEXCLAMATION|MB_OK);
        break;
    }
    m_DQR.SetLastMessageID(0);
    SetInterruptMode(false);

    // Signal thread end
    singleLock.Unlock();
    Beep(700,100);
    return success; // normal termination
}
/*****
*
*   Get thread priority from DICOM message priority
*
*****/
int DQRControl::GetThreadPriority()
{
    if(m_DQR.GetPriority() == ServiceClass::LowPriority)
        return THREAD_PRIORITY_BELOW_NORMAL;
    if(m_DQR.GetPriority() == ServiceClass::HighPriority)
        return THREAD_PRIORITY_ABOVE_NORMAL;
    return THREAD_PRIORITY_NORMAL;
}
/*****
*
*   Determine IP address of the local PC
*
*****/
bool DQRControl::GetLocalIP(CString& sname, BYTE& ip1,BYTE& ip2,
                           BYTE& ip3,BYTE& ip4)
{
    bool success = false;
    WORD wVersionRequested;
    WSADATA wsaData;
    char name[255];
    CString ip;
    PHOSTENT hostinfo;
    wVersionRequested = MAKEWORD( 2, 0 );
    sname="";
    if ( WSASStartup( wVersionRequested, &wsaData ) == 0 )
    {
        if( gethostname ( name, sizeof(name) ) == 0 )
        {
            if((hostinfo = gethostbyname(name)) != NULL)
            {
                ip = inet_ntoa (*(struct in_addr *)*hostinfo->h_addr_list);
                sscanf((char*) (LPCSTR)ip,"%u.%u.%u.%u",&ip1,&ip2,&ip3,&ip4);
                success = true;
                sname = CString(name);
                sname.TrimRight(); sname.TrimLeft();
                sname.MakeUpper();
                if(sname=="")      sname="This_PC";
            }
        }
        WSACleanup();
    }
    if(!success) {    ip1=127; ip2=0; ip3=0; ip4=1; }
    return success;
}
/*****
*
*   Server threads
*
*****/

```

```

*****/
UINT DQRControl::StartQRServer(LPVOID index)
{
    UINT ae_index = (UINT)index;
    ApplicationEntityList *ael = (ApplicationEntityList*)
        (&theApp.app_DataBase.db_AEList);
    if(ae_index >= ael->GetSize())
    {
        AfxEndThread(0);    return 0;
    }
    ael->Get(ae_index).SetPartnerTitle(ael->GetLocalAE().ae_Title);
    Server srv(&theApp.app_DataBase, &theApp.app_ServerLog);
    MarkAEServerStatuses(ael, ae_index, true);
    srv.RunServer(&(ael->Get(ae_index)), false); // Multiple threads for CCancel ?
    MarkAEServerStatuses(ael, ae_index, false);
    AfxEndThread(1);
    return 1;
}

void DQRControl::MarkAEServerStatuses(ApplicationEntityList *ael, UINT ae_index,
                                     bool status)
{
    if(!ael)    return;
    if(ae_index >= ael->GetSize())    return;
    int nPort = ael->Get(ae_index).ae_PortServer;
    ael->SetServedStatus(nPort, status);
    if(!status)
    {
        CString info;    info.Format("Server at port %d abnormally terminated.\n"
                                     "Please restart the application.", nPort);
        AfxMessageBox(info, MB_ICONINFORMATION|MB_OK);
    }
}

bool DQRControl::StartAEServer(UINT ae_index)
{
    if(!m_AEarray)    return false;
    if(ae_index >= m_AEarray->GetSize())    return false;
    if(!m_AEarray->Get(ae_index).ae_Served) // Need to start server
    {
        // Launch server thread
        if(AfxBeginThread(StartQRServer, (LPVOID)ae_index,
            THREAD_PRIORITY_BELOW_NORMAL, 10000) == NULL)
        {
            return false;
        }
        else // just wait for the server thread to start
        {
            while(m_AEarray->Get(ae_index).ae_Served != true) Sleep(100);
        }
    }
    return true;
}

bool DQRControl::StartAllServers()
{
    if(!m_AEarray)    return false;
    UINT failedCount=0;
    UINT nAEs = m_AEarray->GetSize();
    for(UINT n=0; n< nAEs; n++)
    {
        ::ShowProgress((100*(n+1))/nAEs, "Initializing archive servers");
        if(!StartAEServer(n))    failedCount ++;
        Sleep(10);
    }
    ::ShowProgress(0,0);
    if(failedCount>0)    AfxMessageBox("Failed to launch server(s) for some AEs");
    return (failedCount==0);
}

```

```

/*****

```

```

*
*   Function to be called from m_DQR and ServiceClass
*
*****/
int DQRControl::DQRCtrlCallbackFilter(void *dqrctrl, UINT stepnum, UINT id)
{
    if(!dqrctrl)    return 0;
    DQRControl* d;
    try { d = (DQRControl*)dqrctrl; }
    catch(...) { return 0; }
    CString progress;
    if(stepnum == CallBackObject::m_CBConnectingToAE)
    {
        progress = "Connecting to archive ...";
    }
    else if(stepnum == CallBackObject::m_CBConnectionFailed)
    {
        progress = "Connection failed ...";
        d->m_AnimNetwork.Close();
    }
    else if(stepnum == CallBackObject::m_CBSendingRequest)
    {
        progress = "Sending request ...";
        d->m_AnimNetwork.Close();
        d->m_AnimNetwork.Open(IDR_AVI_SEND);
    }
    else if(stepnum == CallBackObject::m_CBGettingResponse)
    {
        progress = "Getting response ...";
        d->m_AnimNetwork.Close();
        d->m_AnimNetwork.Open(IDR_AVI_RECEIVE);
        d->EnableCancel();
    }
    else if(stepnum == CallBackObject::m_CBResponseReceived)
    {
        progress = "Response received";
        d->m_AnimNetwork.Close();
        d->m_AnimNetwork.ShowWindow(SW_HIDE);
        d->m_ButtonCancel.ShowWindow(SW_HIDE);
    }
    else if(stepnum == CallBackObject::m_CBCancelSent)
    {
        progress = "Cancel request sent";
    }
    else if(stepnum == CallBackObject::m_CBDQRTaskSchedule)
    {
        if(d->m_PromptForTaskScheduler)
        {
            d->m_TaskScheduler.RunScheduler(*(DQRTask*)id, true);
        }
        return 1;
    }
    else {}
    if(progress!="")
    {
        d->GetDlgItem(IDC_STATIC_PROGRESS)->ShowWindow(SW_SHOW);
        d->GetDlgItem(IDC_STATIC_PROGRESS)->SetWindowText(progress);
    }
    return d->m_DQR.DQRcallbackFilter(stepnum,id);
}

/*****
*
*   Drug and drop support
*
*****/
void DQRControl::OnBeginDragListResults(NMHDR* pNMHDR, LRESULT* pResult)
{
    NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
    theApp.app_QueryRetrieve.OnBeginDrag(&(this->m_ResultsList), pNMHDR);
    *pResult = 0;
}

void DQRControl::DropOn(CWnd *win)

```

```

{
    if(!win) return;
    if(win->GetParent() == AfxGetMainWnd() ||
        (win->GetTopLevelFrame() == AfxGetMainWnd() && win->GetParent() != this &&
         win->GetParentFrame() != this->GetParentFrame()))
    {
        OnButtonGet();
        return;
    }
    int nid = win->GetDlgCtrlID();
    if(nid==IDC_LIST_RESULTS) // drop on results list
    {
        CWnd* par = win->GetParent();
        if(!par || par == (CWnd*)this) return;
        DQRControl* destin = NULL;
        try
        {
            destin = (DQRControl*)par;
            if(!destin) return;
            UINT nAEFrom = m_ArchiveList.GetCurSel();
            UINT nAETTo = destin->m_ArchiveList.GetCurSel();
            if(nAEFrom == nAETTo)
            {
                AfxMessageBox("You cannot copy to the same archive !");
                return;
            }
            m_MoveArchiveList.SetCurSel(nAETTo);
            UpdateData(FALSE);
            OnButtonMoveTo();
        }
        catch(...) {}
        return;
    }
}

void DQRControl::DropOn(CWnd *wdrag, CWnd *wdrop)
{
    if(wdrag != (CWnd*)(&m_ResultsList)) return;
    DropOn(wdrop);
}

*****
* Column sorts
*****
void DQRControl::OnHeaderClicked(NMHDR *pNMHDR, LRESULT *pResult)
{
    HD_NOTIFY *phdn = (HD_NOTIFY *) pNMHDR;
    if( phdn->iButton == 0 && m_ResultsList.GetItemCnt()>2)
    {
        // User clicked on header using left mouse button
        if( phdn->iItem == m_nSortColumn )
            m_bSortInIncreasingOrder = !m_bSortInIncreasingOrder;
        else
            m_bSortInIncreasingOrder = true;
        m_nSortColumn = phdn->iItem;
        m_ResultsList.SortItems(CompareItems, (DWORD)this);
    }
    *pResult = 0;
}

int CALLBACK DQRControl::CompareItems(LPARAM lParam1, LPARAM lParam2,
                                       LPARAM lParamSort)
{
    int comp=0;
    DQRControl* pDQRCtrl = (DQRControl*)lParamSort;
    if(!pDQRCtrl) return 0;
    // Check if we work with the parent item
    if((DWORD)lParam1 == m_ParentListItemData) return -1;
    if((DWORD)lParam2 == m_ParentListItemData) return 1;
    // Find item index from item data
    LVFINDINFO info;
    info.flags = LVFI_PARAM;
    info.lParam = lParam1;
    int ind1 = pDQRCtrl->m_ResultsList.FindItem(&info);

```

```

if(ind1 == -1) return 0;
info.flags = LVFI_PARAM; info.lParam = lParam2;
int ind2 = pDQRCtrl->m_ResultsList.FindItem(&info);
if(ind2 == -1) return 0;
// Compare the items
int col = pDQRCtrl->m_nSortColumn;
CString s1 = pDQRCtrl->m_ResultsList.GetItemText(ind1, col);
if(s1=="*" || s1=="?" || s1==" " || s1=="") return 1;
CString s2 = pDQRCtrl->m_ResultsList.GetItemText(ind2, col);
if(s2=="*" || s2=="?" || s2==" " || s2=="") return -1;
if( col == m_ColumnBDate || col == m_ColumnBTime ||
    col == m_ColumnSDate || col == m_ColumnSTime ) // dates
{
    COleDateTime dt1, dt2;
    dt1.ParseDateTime(s1);
    if(dt1.GetStatus() != COleDateTime::valid) return 1;
    dt2.ParseDateTime(s2);
    if(dt2.GetStatus() != COleDateTime::valid) return -1;
    if(dt1<dt2) comp=-1;
    else
    {
        if(dt1>dt2) comp=1;
        else comp=0;
    }
}
else if (col == m_ColumnSImgNum) // numbers
{
    int n1 = atoi(s1);
    int n2 = atoi(s2);
    if(n1<n2) comp=-1;
    else
    {
        if(n1>n2) comp=1;
        else comp=0;
    }
}
else // strings
{
    comp = s1.CompareNoCase(s2);
}
if(!pDQRCtrl->m_bSortInIncreasingOrder) comp = -comp;
return comp;

```

```

*****
* Task-processing threads
*
*****/

```

```

UINT DQRControl::RunTaskQueueThread(LPVOID pCtrl)
{

```

```

    if(!pCtrl) return (UINT)FALSE;
    DQRControl* pDQRC = (DQRControl*) pCtrl;
    while(true) // task-processing loop
    {
        pDQRC->UpdateTaskView(pDQRC->m_DQR.ExecuteNextTask());
        Sleep(1000);
    }
}

```

```

bool DQRControl::StartTaskQueue()
{

```

```

    if(!m_UseTaskQueue) return true; // no queues
    return (AfxBeginThread(RunTaskQueueThread, this,
        THREAD_PRIORITY_LOWEST, m_ClientStackSize) != NULL);
}

```

```

void DQRControl::UpdateTaskView(bool reload_list)
{

```

```

    if(reload_list) m_TaskView.LoadTasksList();
    m_TaskView.UpdateClock();
}

```

```

/*****
*
* Persistent storage
*
*****/
void DQRControl::SerializeDQRControl(bool is_loading)
{
    CString fname = theApp.app_DirectoryData;
    if(m_LocalOnly) return; //fname += CString("\\locTasks.dat");
    else fname += CString("\\remTasks.dat");
    FILE* fp;
    fp = fopen((char*)(LPCSTR)fname, is_loading ? "r" : "w");
    if(!fp) return;
    ::Serializebool(fp, m_PromptForTaskScheduler, is_loading);
    if(m_UseTaskQueue)
    {
        if( !is_loading )
        {
            if(m_DQR.GetTasksSize() > 0 &&
                AfxMessageBox("Save your queued queries?",
                    MB_ICONQUESTION | MB_YESNO) == IDYES )
            {
                m_DQR.SerializeDQR(fp, is_loading);
            }
        }
        else m_DQR.SerializeDQR(fp, is_loading);
    }
    fclose(fp);
}

/*****
*
* Task Scheduler
*
*****/
void DQRControl::SetTaskSchedulerPrompt(bool enable)
{
    m_PromptForTaskScheduler = enable;
    // No prompts on local or without queues
    if(m_LocalOnly || !m_UseTaskQueue) m_PromptForTaskScheduler = false;
}

bool DQRControl::GetTaskSchedulerPrompt()
{
    return m_PromptForTaskScheduler;
}

```



```

#if !defined(AFX_DQRTASKVIEW_H_INCLUDED_)
#define AFX_DQRTASKVIEW_H_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// dqrtaskview.h : header file
//

////////////////////////////////////
// DQRTaskView dialog

class DQRTaskView : public CDialog
{
// Construction
public:
    void                UpdateClock();
    void                LoadTasksList();
    bool                AttachDQR(DQR* pDDR);
    DQRTaskView(CWnd* pParent = NULL);    // standard constructor
    ~DQRTaskView() { m_ImageList.DeleteImageList(); };

// Dialog Data
   //{{AFX_DATA(DQRTaskView)
    enum { IDD = IDD_DIALOG_DQRTASKVIEW };
    CListCtrl    m_TasksList;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(DQRTaskView)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
   //{{AFX_MSG(DQRTaskView)
    virtual BOOL OnInitDialog();
    afx_msg void OnDelete();
    afx_msg void OnRescheduleTask();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    const static int    m_ColumnnTasks, m_ColumnArchive, m_ColumnData,
                        m_ColumnAttempts, m_ColumnSchedule;
    CImageList          m_ImageList;
    DQR*                m_pDQR;
};

////////////////////////////////////
// DQRTaskSchedule dialog

class DQRTaskSchedule : public CDialog
{
// Construction
public:
    void                RunScheduler(DQRTask& t, bool prompt);
    void                ScheduleTask(DQRTask& t);
    DQRTaskSchedule(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(DQRTaskSchedule)
    enum { IDD = IDD_DIALOG_DQRTASKVIEW_SCHEDULE };
    CComboBox    m_ComboTo;
    CComboBox    m_ComboFrom;
    CTime        m_EndDate;
    CTime        m_EndTime;
    CTime        m_StartDate;
    CTime        m_StartTime;
    //}}AFX_DATA

```

```

UINT    m_Attempts;
//}}AFX_DATA

- // Overrides
- // ClassWizard generated virtual function overrides
- //{{AFX_VIRTUAL(DQRTaskSchedule)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(DQRTaskSchedule)
afx_msg void OnSelChangeComboFrom();
virtual BOOL OnInitDialog();
afx_msg void OnSelChangeComboTo();
virtual void OnOK();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
private:
    BYTE        m_FromIndex, m_ToIndex;
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DQRTASKVIEW_H_INCLUDED_)

```

```
// dqrtaskview.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "..\DCM.h"
#include "dqrtaskview.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// DQRTaskView dialog
```

```
const int DQRTaskView::m_ColumnTasks = 0;
const int DQRTaskView::m_ColumnArchive = 1;
const int DQRTaskView::m_ColumnData = 2;
const int DQRTaskView::m_ColumnAttempts = 3;
const int DQRTaskView::m_ColumnSchedule = 4;
```

```
DQRTaskView::DQRTaskView(CWnd* pParent /*=NULL*/)
: CDialog(DQRTaskView::IDD, pParent)
```

```
{
    //{{AFX_DATA_INIT(DQRTaskView)
    // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
    m_pDQR = NULL;
}
```

```
void DQRTaskView::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(DQRTaskView)
    DDX_Control(pDX, IDC_LIST_TASKS, m_TasksList);
    //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(DQRTaskView, CDialog)
    //{{AFX_MSG_MAP(DQRTaskView)
    ON_BN_CLICKED(ID_DELETE, OnDelete)
    ON_BN_CLICKED(ID_RESCHEDULE, OnRescheduleTask)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// DQRTaskView message handlers
```

```
bool DQRTaskView::AttachDQR(DQR* pDQR)
```

```
{
    if(!pDQR) return false;
    m_pDQR = pDQR;
    return true;
}
```

```
BOOL DQRTaskView::OnInitDialog()
```

```
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    /* SET m_TasksList : */
    // 1. Load icons
    if(m_ImageList.m_hImageList==NULL)
    {
        if(!m_ImageList.Create(16,17,ILC_COLOR4,3,1))
        {
            return FALSE;
        }
        m_ImageList.Add(theApp.LoadIcon(IDI_TASK_DOWNLOAD));
        m_ImageList.Add(theApp.LoadIcon(IDI_TASK_SCHEDULE));
    }
    m_TasksList.SetImageList(&m_ImageList,LVSIL_SMALL);
}
```

```

// 2. Load columns
m_TasksList.InsertColumn(m_ColumnTasks,"Task Status", LVCFMT_CENTER,110,0);
m_TasksList.InsertColumn(m_ColumnArchive,"Archive", LVCFMT_CENTER,110,0);
m_TasksList.InsertColumn(m_ColumnData,"Data", LVCFMT_CENTER,190,0);
m_TasksList.InsertColumn(m_ColumnAttempts,"Attempts", LVCFMT_CENTER,60,0);
m_TasksList.InsertColumn(m_ColumnSchedule,"Schedule", LVCFMT_CENTER,80,0);
m_TasksList.SetColumnWidth(m_ColumnSchedule,LVSCW_AUTOSIZE_USEHEADER);
// 3. Force entire row selection
m_TasksList.SetExtendedStyle(LVS_EX_FULLROWSELECT | LVS_EX_SUBITEMIMAGES
                             | LVS_EX_GRIDLINES);

// Display current tasks
LoadTasksList();

return TRUE; // return TRUE unless you set the focus to a control
             // EXCEPTION: OCX Property Pages should return FALSE
}

/*****
*
*   (Re)Load the list of tasks
*
*****/
void DQRTaskView::LoadTasksList()
{
    if(!m_pDQR || !m_TasksList.GetSafeHwnd()) return;
    m_TasksList.DeleteAllItems();
    char    s[128];
    int     nItem, nItemType;
    CString sItemType;
    DQRTask t;

    for(int n=m_pDQR->GetTasksSize()-1; n>=0; n--)
    {
        if(!m_pDQR->GetTask(n,t)) continue;
        // Find out the item type
        if(t.CanExecuteNow())
        {
            nItemType = 0;  sItemType = " In Progress...";
        }
        else
        {
            if(t.CanExecuteLater())
            {
                nItemType = 1;  sItemType = " On Schedule";
            }
            else
            {
                nItemType = 0;  sItemType = " Finishing ...";
            }
        }

        nItem = m_TasksList.InsertItem(n, sItemType, nItemType);
        m_pDQR->GetAELocation(t.m_nAE, s);
        m_TasksList.SetItem(nItem, m_ColumnArchive, LVIF_TEXT, s, 0, 0, 0, 0);
        sprintf(s,"%s, %s", t.m_DRdata.GetPatientName(), t.m_DRdata.GetPatientID());
        m_TasksList.SetItem(nItem, m_ColumnData, LVIF_TEXT, s, 0, 0, 0, 0);
        sprintf(s, "%d", t.GetExec());
        m_TasksList.SetItem(nItem, m_ColumnAttempts, LVIF_TEXT, s, 0, 0, 0, 0);
        t.FormatScheduleString(s,127);
        m_TasksList.SetItem(nItem, m_ColumnSchedule, LVIF_TEXT, s, 0, 0, 0, 0);
        m_TasksList.SetItemData(nItem,t.GetID());
    }
    // Prevent horizontal scroll
    m_TasksList.SetColumnWidth(m_ColumnSchedule,LVSCW_AUTOSIZE_USEHEADER);
    UpdateClock();
}

/*****
*
*   Update clock on the dialog
*
*****/
void DQRTaskView::UpdateClock()
{

```

```

    if(!GetSafeHwnd()) return;
    // Show current date and time
    CTime tm = CTime::GetCurrentTime();
    GetDlgItem(IDC_CURRENT)
        ->SetWindowText(tm.Format("%A, %d %B %Y, %Hh %Mm %Ss"));
}
/*****
*
*   Delete selected tasks
*
*****/
void DQRTaskView::OnDelete()
{
    if(!m_pDQR) return;
    POSITION pos = m_TasksList.GetFirstSelectedItemPosition();
    if (pos == NULL) return; // nothing selected
    int sel=-1;
    UINT taskID;
    while(pos)
    {
        sel = m_TasksList.GetNextSelectedItem(pos);
        if(sel<0) continue;
        taskID = m_TasksList.GetItemData(sel);
        m_pDQR->RemoveTaskID(taskID);
    }
    LoadTasksList();
}

/*****
*
*   Reschedule selected tasks
*
*****/
void DQRTaskView::OnRescheduleTask()
{
    if(!m_pDQR) return;
    POSITION pos = m_TasksList.GetFirstSelectedItemPosition();
    if (pos == NULL) return; // nothing selected
    int sel=-1;
    UINT taskID;
    DQRTask* pTask;
    DQRTaskSchedule dts;
    if(dts.DoModal() != IDOK) return;
    while(pos)
    {
        sel = m_TasksList.GetNextSelectedItem(pos);
        if(sel<0) continue;
        taskID = m_TasksList.GetItemData(sel);
        pTask = m_pDQR->GetTaskPtrFromID(taskID);
        if(pTask) dts.ScheduleTask(*pTask);
    }
    LoadTasksList();
}

// DQRTaskSchedule dialog

DQRTaskSchedule::DQRTaskSchedule(CWnd* pParent /*=NULL*/)
: CDialog(DQRTaskSchedule::IDD, pParent)
{
    //{{AFX_DATA_INIT(DQRTaskSchedule)
    m_EndDate = CTime::GetCurrentTime();
    m_EndTime = CTime::GetCurrentTime();
    m_StartDate = CTime::GetCurrentTime();
    m_StartTime = CTime::GetCurrentTime();
    m_Attempts = 1;
    //}}AFX_DATA_INIT

```

```

    m_FromIndex=m_ToIndex=0;
}

void DQRTaskSchedule::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(DQRTaskSchedule)
    DDX_Control(pDX, IDC_COMBO_TO, m_ComboTo);
    DDX_Control(pDX, IDC_COMBO_FROM, m_ComboFrom);
    DDX_DateTimeCtrl(pDX, IDC_END_DATE, m_EndDate);
    DDX_DateTimeCtrl(pDX, IDC_END_TIME, m_EndTime);
    DDX_DateTimeCtrl(pDX, IDC_START_DATE, m_StartDate);
    DDX_DateTimeCtrl(pDX, IDC_START_TIME, m_StartTime);
    DDX_Text(pDX, IDC_ATTEMPTS, m_Attempts);
    DDV_MinMaxUInt(pDX, m_Attempts, 1, 5);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(DQRTaskSchedule, CDialog)
    //{{AFX_MSG_MAP(DQRTaskSchedule)
    ON_CBN_SELCHANGE(IDC_COMBO_FROM, OnSelChangeComboFrom)
    ON_CBN_SELCHANGE(IDC_COMBO_TO, OnSelChangeComboTo)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// DQRTaskSchedule message handlers
void DQRTaskSchedule::OnSelChangeComboFrom()
{
    m_FromIndex = m_ComboFrom.GetCurSel();
    CString txt;
    m_ComboFrom.GetWindowText(txt);
    int sw = (txt=="specific time" ? SW_NORMAL : SW_HIDE);
    GetDlgItem(IDC_START_DATE)->ShowWindow(sw);
    GetDlgItem(IDC_START_TIME)->ShowWindow(sw);
}

void DQRTaskSchedule::OnSelChangeComboTo()
{
    m_ToIndex = m_ComboTo.GetCurSel();
    CString txt;
    m_ComboTo.GetWindowText(txt);
    int sw = (txt=="specific time" ? SW_NORMAL : SW_HIDE);
    GetDlgItem(IDC_END_DATE)->ShowWindow(sw);
    GetDlgItem(IDC_END_TIME)->ShowWindow(sw);
}

BOOL DQRTaskSchedule::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_ComboFrom.SetCurSel(m_FromIndex); OnSelChangeComboFrom();
    m_ComboTo.SetCurSel(m_ToIndex); OnSelChangeComboTo();

    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void DQRTaskSchedule::OnOK()
{
    UpdateData(TRUE);
    CDialog::OnOK();
}

/*****
*
* Write schedule information into a task
*
*****/
void DQRTaskSchedule::ScheduleTask(DQRTask &t)
{
    CTime tm;
    // 1. Find out start date/time

```

```

        // No choice was made from the listbox,
        // and the edit control was empty
        m_ListIndex=0;
        m_AEComboList.SetCurSel(0);
        UpdateAllFields(true);
    }
    else if((n=m_AEComboList.FindStringExact(-1,info)) != CB_ERR)
    {
        // The edited string already exists
        m_ListIndex=n;
        m_AEComboList.SetCurSel(n);
        UpdateAllFields(true);
    }
    else
    {
        // Totally new string was entered
        m_AEarray->Get(m_ListIndex).SetLocation((char*)(LPCSTR)info);
        m_AEComboList.SetCurSel(m_ListIndex);
    }
}

/*****
 *
 * Buttons
 *
 *****/
void AEOptions_Dialog::OnOK()
{
    // TODO: Add extra validation here
    UpdateAllFields(false);
    OnCloseupComboAeList();
    CDialog::OnOK();
}

void AEOptions_Dialog::OnAENew()
{
    ApplicationEntity a("<New AE Title>", 255,255,255,255);
    m_AEarray->Add(a);
    ResetAEList(m_AEarray->GetUpperBound());
}

void AEOptions_Dialog::OnAeClone()
{
    UpdateAllFields(false);
    ApplicationEntity a = m_AEarray->Get(m_ListIndex);
    CString s; s.Format("%s_Copy",a.ae_Title);
    s = s.Left(min(s.GetLength(),16));
    a.SetTitle((char*)(LPCSTR)(s));
    m_AEarray->Add(a);
    ResetAEList(m_AEarray->GetUpperBound());
}

void AEOptions_Dialog::OnAEDelete()
{
    int ind = m_AEComboList.GetCurSel();
    if(ind == (int)m_AEarray->GetLocalIndex())
    {
        AfxMessageBox("You cannot delete local AE",
            MB_ICONEXCLAMATION|MB_OK);
        return;
    }
    if(ind>=(int)(m_AEarray->GetSize()) || ind<0) return;
    m_AEarray->RemoveAt(ind);
    if(ind>0) ind--;
    ResetAEList(ind);
}

/*****
 *
 * Totally reset AE list
 *
 *****/

```

```

void AEOptions_Dialog::ResetAEList(int new_selection)
{
    m_AEComboList.ResetContent();
    CString loc;
    for(UINT i=0; i<m_AEarray->GetSize(); i++)
    {
        loc=CString(m_AEarray->Get(i).ae_Location);
        if(i==m_AEarray->GetLocalIndex() && loc.Find("<Local>")<0)
        {
            loc=CString("<Local> ") + loc;
        }
        m_AEComboList.InsertString(i, loc);
    }
    if(new_selection>=(int)(m_AEarray->GetSize()) || new_selection<0)    new_selection=0;
    m_ListIndex=new_selection;
    m_AEComboList.SetCurSel(new_selection);
    UpdateAllFields(true);
}

```

```

/*****
 *
 *   Update selection index
 *
 *****/
void AEOptions_Dialog::OnSelchangeComboAeList()
{

```

```

    m_ListIndex=m_AEComboList.GetCurSel();
}

```



```

#if !defined(AFX_DQRCONTROL_H_INCLUDED_)
#define AFX_DQRCONTROL_H_INCLUDED_

#include "..\Controls.h"      // Added by ClassView
#include "dqrtaskview.h"     // Added by ClassView
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// dqrcontrol.h : header file
//

/////////////////////////////////////////////////////////////////
// DQRSearch dialog

class DQRSearch : public CDialog
{
// Construction
public:
    void                WriteIntoDICOMRecord(DICOMRecord& dr);
    DateTimeSegment*    GetBirthDatePtr();
    DQRSearch(CWnd* pParent = NULL);    // standard constructor
// Dialog Data
    //{AFX_DATA(DQRSearch)
    enum { IDD = IDD_DIALOG_DQRSEARCH };
    CString m_PatientID;
    CString m_PatientName;
    CString m_AccessionNumber;
    CString m_StudyID;
    CString m_StudyInstUID;
    CString m_Modality;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(DQRSearch)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(DQRSearch)
    virtual BOOL OnInitDialog();
    virtual void OnOK();
    afx_msg void OnButtonAdvancedOrBasic();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    bool                m_Advanced;
    DateTimeControl     m_BirthDateControl, m_BirthTimeControl,
                        m_StudyDateControl, m_StudyTimeControl;

    void                SizeDialogArea();
};

/////////////////////////////////////////////////////////////////
// DQRControl dialog

class DQRControl : public CDialog
{
// Construction
public:
    void                SetTaskSchedulerPrompt(bool enable);
    void                LoadArchiveList(UINT selection = 0);
    void                DropOn(CWnd* wdrag, CWnd* wdrop);
    void                UpdateAfterFind(bool find_success);
    void                SetInterruptMode(bool interrupt);
    void                SetPriority(BYTE priority) { m_DQR.SetPriority(priority); };
    void                ShowTaskView() { if (HasTaskQueue()) m_TaskView.DoModal(); };
    bool                DisplayOverControl(int controlID, CWnd *parent);

```

```

    bool        GetTaskSchedulerPrompt();
    bool        CreateDQRControl(DICOMViewLog* ptrClientLog, DICOMDatabase* ptrDB,
                                bool local_only);
    bool        HasTaskQueue()          { return m_UseTaskQueue ; }
    BYTE        GetPriority()            { return m_DQR.GetPriority(); };

    DQRControl(CWnd* pParent = NULL);    // constructor
    ~DQRControl();                      // destructor

// Dialog Data
//{{AFX_DATA(DQRControl)
enum { IDD = IDD_DIALOG_DQRCONTROL };
CAnimateCtrl    m_AnimNetwork;
CButton         m_ButtonSearch;
CButton         m_ButtonCancel;
CButton         m_ButtonMoveTo;
CButton         m_ButtonGet;
CListCtrl       m_ResultsList;
CComboBox       m_MoveArchiveList;
CComboBox       m_ArchiveList;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(DQRControl)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
    void OnHeaderClicked(NMHDR* pNMHDR, LRESULT* pResult);
    // Generated message map functions
//{{AFX_MSG(DQRControl)
    virtual BOOL OnInitDialog();
    afx_msg void OnButtonStartSearch();
    afx_msg void OnButtonGet();
    afx_msg void OnButtonMoveTo();
    afx_msg void OnDoubleClickListResults(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnSelChangeArchiveList();
    afx_msg void OnSelChangeMoveArchiveList();
    afx_msg void OnButtonCancel();
    afx_msg void OnBeginDragListResults(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnButtonDelete();
//}}AFX_MSG
    afx_msg void OnOk() {} // ignore when Enter is pressed
    afx_msg bool OnClickListResults(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnRightClickListResults(NMHDR* pNMHDR, LRESULT* pResult);
    DECLARE_MESSAGE_MAP()

private:
    bool        m_LocalOnly, m_UseTaskQueue, m_PromptForTaskScheduler;
    bool        m_PreloadData;
    bool        m_bSortInIncreasingOrder;
    char        m_MoveDestinationAE[20];
    int         m_nSortColumn;
    int         m_ResultsListSelection[5];
    int         m_DOBindex;
    const static int m_ColumnPname, m_ColumnPid, m_ColumnAnum,
                    m_ColumnMod, m_ColumnSDate, m_ColumnSTime,
                    m_ColumnBDate, m_ColumnBTime, m_ColumnsImgNum;
    const static UINT m_ClientStackSize;
    const static DWORD m_ParentListItemData;
    HWND        m_HWND;
    CCriticalSection m_RunClientThread_CritSection;
    ApplicationEntityList* m_AEarray;
    CImageList    m_ImageList;
    DQRSearch     m_SearchDialog;
    DQR           m_DQR;
    DQRTaskView   m_TaskView;
    DQRTaskSchedule m_TaskScheduler;
    enum
    {
        echo,

```

```

    find, find_root, find_previous, find_next,
    get_index, move_index
}
    m_RequestedClientService;

void        SerializeDQRControl(bool is_loading);
void        EnableCancel();
void        UpdateTaskView(bool reload_list);
void        DropOn(CWnd* win);
void        TestArchiveConnection();
void        FindAll();
static void MarkAEServerStatuses(ApplicationEntityList *ael,
                                UINT ae_index, bool status);

bool        LoadFoundList(bool erase);
bool        StartAllServers();
bool        StartAEServer(UINT ae_index);
bool        StartTaskQueue();
bool        RunClientThread();
static bool GetLocalIP(CString& sname, BYTE& ip1, BYTE& ip2,
                       BYTE& ip3, BYTE& ip4);

BOOL        UpdateData( BOOL bSaveAndValidate=TRUE);
int         GetThreadPriority();
int         GetResultsListSelection(int& sel, int& dcm_index);
static int  DQRCtrlCallbackFilter(void* dqrctrl, UINT stepnum=0,
                                UINT id=0);

static int CALLBACK CompareItems(LPPARAM lParam1,
                                LPARAM lParam2, LPARAM lParamSort);

static UINT StartQRServer(LPVOID index);
static UINT StartQRClient(LPVOID ptrDQRControl);
static UINT RunTaskQueueThread(LPVOID pCtrl);
CString     GetLevelPrompt(bool higher);
}

#ifdef _AFX_DQRCONTROL_H_INCLUDED_

```

```
// dqrcontrol.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "..\DCM.h"
#include "dqrcontrol.h"
#include "Server.h"
#include <process.h>
```

```
#ifdef _DEBUG
```

```
#define new DEBUG_NEW
```

```
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
```

```
#endif
```

```
////////////////////////////////////
// DQRSearch dialog
```

```
DQRSearch::DQRSearch(CWnd* pParent /*=NULL*/)
: CDialog(DQRSearch::IDD, pParent)
```

```
{
    //{{AFX_DATA_INIT(DQRSearch)
    m_PatientID = _T("");
    m_PatientName = _T("");
    m_AccessionNumber = _T("");
    m_StudyID = _T("");
    m_StudyInstUID = _T("");
    m_Modality = _T("");
    //}}AFX_DATA_INIT
    m_Advanced = false;
    m_BirthDateControl.SetDateFormat();
    m_BirthDateControl.SetTitle("Birth Date: ");
    m_BirthTimeControl.SetTimeFormat();
    m_BirthTimeControl.SetTitle("Birth Time: ");
    m_StudyDateControl.SetDateFormat();
    m_StudyDateControl.SetTitle("Study Date: ");
    m_StudyTimeControl.SetTimeFormat();
    m_StudyTimeControl.SetTitle("Study Time: ");
}
```

```
void DQRSearch::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(DQRSearch)
    DDX_Text(pDX, IDC_PATIENT_ID, m_PatientID);
    DDX_Text(pDX, IDC_PATIENT_NAME, m_PatientName);
    DDX_Text(pDX, IDC_STUDY_ACCNUM, m_AccessionNumber);
    DDV_MaxChars(pDX, m_AccessionNumber, 16);
    DDX_Text(pDX, IDC_STUDY_ID, m_StudyID);
    DDV_MaxChars(pDX, m_StudyID, 16);
    DDX_Text(pDX, IDC_STUDY_INSTUID, m_StudyInstUID);
    DDV_MaxChars(pDX, m_StudyInstUID, 64);
    DDX_Text(pDX, IDC_MODALITY, m_Modality);
    DDV_MaxChars(pDX, m_Modality, 3);
    //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(DQRSearch, CDialog)
```

```
    //{{AFX_MSG_MAP(DQRSearch)
    ON_BN_CLICKED(IDC_BUTTON_ADV, OnButtonAdvancedOrBasic)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// DQRSearch message handlers
```

```
BOOL DQRSearch::OnInitDialog()
```

```
{
    CDialog::OnInitDialog();
    // Place DateTimeSegment controls
    if(!m_BirthDateControl.DisplayOverControl(IDC_BIRTHDATE, this))
        return FALSE;
    if(!m_BirthTimeControl.DisplayOverControl(IDC_BIRTHTIME, this))
        return FALSE;
}
```

```

    if(!m_StudyDateControl.DisplayOverControl(IDC_STDATE, this))
        return FALSE;
    if(!m_StudyTimeControl.DisplayOverControl(IDC_STTIME, this))
        return FALSE;

    SizeDialogArea();
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

```

```

/*****
 *
 *   Validating input on exit
 *
 *****/
void DQRSearch::OnOK()
{

```

```

    UpdateData(TRUE);
    m_BirthDateControl.UpdateData(TRUE);
    m_BirthTimeControl.UpdateData(TRUE);
    CDialog::OnOK();
}

```

```

/*****
 *
 *   Switch between advanced and basic parameters
 *
 *****/
void DQRSearch::SizeDialogArea()
{

```

```

    CWnd* pWnd = GetDlgItem(IDC_BUTTON_ADV);
    if(!pWnd) return;
    if(!m_Advanced)
    {
        CRect rc, rcb;
        pWnd->GetWindowRect(rcb);
        GetWindowRect(rc);
        //ScreenToClient(rc);
        rc.bottom = rcb.bottom+10;
        MoveWindow(rc);
        pWnd->SetWindowText("Advanced >>");
    }

```

```

    else
    {
        CWnd* pWnd2 = GetDlgItem(IDC_STTIME);
        if(!pWnd2) return;
        CRect rc, rcb;
        pWnd2->GetWindowRect(rcb);
        GetWindowRect(rc);
        //ScreenToClient(rc);
        rc.bottom = rcb.bottom+10;
        MoveWindow(rc);
        pWnd->SetWindowText("Basic <<");
    }
}

```

```

void DQRSearch::OnButtonAdvancedOrBasic()
{
    m_Advanced = !m_Advanced;
    SizeDialogArea();
}

```

```

/*****
 *
 *   Access DateTimeSegments
 *
 *****/
DateTimeSegment* DQRSearch::GetBirthDatePtr()
{
    return &(m_BirthDateControl.GetDateTimeSegment());
}

```

```

/*****
 *
 *   Fill DICOMRecord with the data from this dialog

```

```

*
*****

```

```

void DQRSearch::WriteIntoDICOMRecord(DICOMRecord &dr)

```

```

{
    if(!m_Advanced) // baseline search
    {
        dr.SetRecord( (char*)(LPCSTR)m_PatientID,
            (char*)(LPCSTR)m_PatientName,
            &(m_BirthDateControl.GetDateTimeSegment()),
            &(m_BirthTimeControl.GetDateTimeSegment()),
            NULL,
            NULL, NULL, NULL,
            NULL, NULL,
            NULL, NULL, NULL,
            NULL, NULL, NULL);
    }
    else
    {
        dr.SetRecord( (char*)(LPCSTR)m_PatientID,
            (char*)(LPCSTR)m_PatientName,
            &(m_BirthDateControl.GetDateTimeSegment()),
            &(m_BirthTimeControl.GetDateTimeSegment()),
            (char*)(LPCSTR)m_StudyInstUID,
            (char*)(LPCSTR)m_StudyID,
            (char*)(LPCSTR)m_AccessionNumber, NULL,
            &(m_StudyDateControl.GetDateTimeSegment()),
            &(m_StudyTimeControl.GetDateTimeSegment()),
            NULL, (char*)(LPCSTR)m_Modality, NULL,
            NULL, NULL, NULL);
    }
}

```

```

// DQRControl.cpp : implementation file

```

```

const int DQRControl::m_ColumnPname = 0;
const int DQRControl::m_ColumnPid = 1;
const int DQRControl::m_ColumnBDate = 2;
const int DQRControl::m_ColumnSDate = 3;
const int DQRControl::m_ColumnSTime = 4;
const int DQRControl::m_ColumnMod = 5;
const int DQRControl::m_ColumnSImgNum = 6;
const int DQRControl::m_ColumnAnum = 7;
const int DQRControl::m_ColumnBTime = 8;
const UINT DQRControl::m_ClientStackSize=1000;
const DWORD DQRControl::m_ParentListItemData=999999;

```

```

////////////////////////////////////
// DQRControl dialog

```

```

DQRControl::DQRControl(CWnd* pParent /*=NULL*/)
: CDialog(DQRControl::IDD, pParent)
{
    //{{AFX_DATA_INIT(DQRControl)
    //}}AFX_DATA_INIT
    m_HWND = NULL;
    m_AEarray = NULL;
    m_LocalOnly=false; m_UseTaskQueue=true; // Queue tasks on remote
    m_PromptForTaskScheduler = true; // Prompt for task scheduler on remote
    m_PreloadData = false;
    m_bSortInIncreasingOrder = true;
    m_nSortColumn = 0;
}

DQRControl::~DQRControl()

```

```

{
    m_ImageList.DeleteImageList();
    if(!m_LocalOnly && m_AEarray)
    {
        m_AEarray->SetCurrentIndex(m_DQR.GetCurrentAEIndex());
    }
    SerializedDQRControl(false);
}

void DQRControl::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(DQRControl)
    DDX_Control(pDX, IDC_ANIMATE_NETWORKING, m_AnimNetwork);
    DDX_Control(pDX, IDC_BUTTON_START_SEARCH, m_ButtonSearch);
    DDX_Control(pDX, IDC_BUTTON_CANCEL, m_ButtonCancel);
    DDX_Control(pDX, IDC_BUTTON_MOVE_TO, m_ButtonMoveTo);
    DDX_Control(pDX, IDC_BUTTON_GET, m_ButtonGet);
    DDX_Control(pDX, IDC_LIST_RESULTS, m_ResultsList);
    DDX_Control(pDX, IDC_COMBO_MOVEARCHIVE, m_MoveArchiveList);
    DDX_Control(pDX, IDC_COMBO_ARCHIVE, m_ArchiveList);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(DQRControl, CDialog)
    //{{AFX_MSG_MAP(DQRControl)
    ON_BN_CLICKED(IDC_BUTTON_START_SEARCH, OnButtonStartSearch)
    ON_BN_CLICKED(IDC_BUTTON_GET, OnButtonGet)
    ON_BN_CLICKED(IDC_BUTTON_MOVE_TO, OnButtonMoveTo)
    ON_NOTIFY(NM_DBLCLK, IDC_LIST_RESULTS, OnDoubleClickListResults)
    ON_CBN_SELCHANGE(IDC_COMBO_ARCHIVE, OnSelChangeArchiveList)
    ON_CBN_SELCHANGE(IDC_COMBO_MOVEARCHIVE, OnSelChangeMoveArchiveList)
    ON_BN_CLICKED(IDC_BUTTON_CANCEL, OnButtonCancel)
    ON_NOTIFY(LVN_BEGINDRAG, IDC_LIST_RESULTS, OnBeginDragListResults)
    ON_NOTIFY(NM_CLICK, IDC_LIST_RESULTS, OnClickListResults)
    ON_NOTIFY(NM_RCLICK, IDC_LIST_RESULTS, OnRightClickListResults)
    ON_BN_CLICKED(IDC_BUTTON_DELETE, OnButtonDelete)
    //}}AFX_MSG_MAP
    ON_COMMAND(IDOK, OnOk)
    ON_COMMAND(IDCANCEL, OnOk)
    ON_NOTIFY(HDN_ITEMCLICKA, 0, OnHeaderClicked) // column sort
    ON_NOTIFY(HDN_ITEMCLICKW, 0, OnHeaderClicked) // column sort
END_MESSAGE_MAP()

//DQRControl message handlers

/*****
 *
 *   Display the control
 *
 *****/
bool DQRControl::DisplayOverControl(int controlID, CWnd *parent)
{
    if(!parent || !controlID) return false;
    CWnd* pWnd = parent->GetDlgItem(controlID);
    if (pWnd)
    {
        CRect rc;
        pWnd->GetWindowRect(rc);
        parent->ScreenToClient(rc);
        this->Create(IDD, parent);
        this->SetWindowPos(pWnd, rc.left, rc.top, 0, 0, SWP_NOSIZE|SWP_SHOWWINDOW);
        return true;
    }
    return false;
}

/*****
 *
 *   Get string for the current QR level
 *
 *****/

```

```

*****/
CString DQRControl::GetLevelPrompt(bool higher)
{
    CString prompt;
    BYTE lev = m_DQR.GetLevel();
    if(!higher)
    {
        if(lev == DICOMRecord::LevelPatient)    prompt = "PATIENT";
        else if(lev == DICOMRecord::LevelStudy) prompt = "STUDY";
        else if(lev == DICOMRecord::LevelSeries) prompt = "SERIES";
        else if(lev == DICOMRecord::LevelImage) prompt = "IMAGE";
        else prompt = "";
    }
    else
    {
        if(m_DQR.IsOnRootLevel())    prompt=" Start new search";
        else if(lev == DICOMRecord::LevelStudy)    prompt = " Go to patient level";
        else if(lev == DICOMRecord::LevelSeries)    prompt = " Go to study level";
        else if(lev == DICOMRecord::LevelImage)    prompt = " Go to series level";
        else prompt = "";
    }
    return prompt;
}

/*****
*
*   (Re)Load the list of archives
*
*****/
void DQRControl::LoadArchiveList(UINT selection /*=0*/)
{
    if(!m_AEarray)    return;
    CString loc;
    m_ArchiveList.ResetContent();    m_MoveArchiveList.ResetContent();
    for(UINT i=0; i<m_AEarray->GetSize(); i++)
    {
        loc=CString(m_AEarray->Get(i).ae_Location);
        if(i==m_AEarray->GetLocalIndex() && loc.Find("<Local>")<0)
        {
            loc = CString("<Local> ") + loc;
        }
        m_ArchiveList.InsertString(i, loc);
        m_MoveArchiveList.InsertString(i, loc);
    }

    if(m_LocalOnly) selection = m_AEarray->GetLocalIndex();
    m_DQR.SetCurrentAEIndex(selection);
    selection=m_DQR.GetCurrentAEIndex();
    m_ArchiveList.SetCurSel(selection);
    if(selection>0) selection=0; else selection=1;
    m_MoveArchiveList.SetCurSel(selection);
    if(m_LocalOnly)
    {
        m_ArchiveList.GetWindowText(loc);
        loc += CString(" archive");
        GetDlgItem(IDC_ARCHIVE_LOCATION)->SetWindowText(loc);
        m_ArchiveList.ShowWindow(SW_HIDE);
    }
    UpdateData(FALSE);
}

/*****
*
*   On new selection in the archive list
*
*****/
void DQRControl::OnSelChangeArchiveList()
{
    if(!m_AEarray && m_LocalOnly)    return;
    UINT n=m_ArchiveList.GetCurSel();
    if(n!=m_DQR.GetCurrentAEIndex())    LoadFoundList(true);
    if(!m_DQR.SetCurrentAEIndex(n))
    {
        UINT m = m_DQR.GetCurrentAEIndex();
        if(m>n)

```



```

    {
        m=0;
        m_DQR.SetCurrentAEIndex(0);
    }
    m_ArchiveList.SetCurSel(m);
    LoadFoundList(true);
}
TestArchiveConnection();
// Make sure current archive is different from "Move to" archive
n=m_ArchiveList.GetCurSel();
if(m_MoveArchiveList.GetCurSel()==(int)n)
{
    if(n>0) n=0; else n=1;
    m_MoveArchiveList.SetCurSel(n);
}
}
void DQRControl::OnSelChangeMoveArchiveList()
{
    if(!m_AEarray) return;
    int n = m_ArchiveList.GetCurSel();
    if(n>=(int)m_AEarray->GetSize())
    {
        LoadArchiveList(); return;
    }
    if(m_MoveArchiveList.GetCurSel() == n)
    {
        CString s;
        m_ArchiveList.GetWindowText(s);
        CString info;
        info.Format("You are currently connected to %s\n"
            " and cannot use it as move destination.", s);
        AfxMessageBox(info, MB_ICONINFORMATION);
        if(n>0) n=n-1; else n=1;
        m_MoveArchiveList.SetCurSel(n);
    }
}

/*****
*
* (Re)Load the list of found DICOM data objects
*
*****/
bool DQRControl::LoadFoundList(bool erase)
{
    if(!GetSafeHwnd()) return false;
    CString label;
    CString lev=GetLevelPrompt(false);
    CString res_lev=CString(" on ")+lev+CString(" level");
    lev.MakeLower();
    m_ButtonGet.EnableWindow(FALSE);
    label = m_LocalOnly ? "Open " : "Download ";
    m_ButtonGet.SetWindowText(label+lev);
    m_ButtonMoveTo.EnableWindow(FALSE);
    label = m_LocalOnly ? "Upload " : "Move ";
    m_ButtonMoveTo.SetWindowText(label+lev+ CString(" to: "));
    m_ResultsList.DeleteAllItems();
    CString res("Results: no matches");
    if(erase)
    {
        m_DQR.ClearFound();
        m_ResultsListSelection[0]=1;
        m_ResultsListSelection[1]=1;
        m_ResultsListSelection[2]=1;
        m_ResultsListSelection[3]=1;
        GetDlgItem(IDC_RESULTS_FRAME)->SetWindowText(res+res_lev);
        UpdateData(FALSE);
        return true;
    }
    int nFound = m_DQR.GetFoundCount();
    m_ResultsList.SetItemCount(nFound+1);
    GetDlgItem(IDC_STATIC_PROGRESS)->SetWindowText(
        nFound>0 ? "Loading data..." : "");
    char s[64];
    int nItem, nI;

```

```

CString str;
DICOMRecord dr;
for(nI=nFound-1; nI>=0; nI--)
{
    m_DQR.GetFoundRecord(dr,nI);
    // Patient name
    str=CString(dr.GetPatientName()); str.Replace('^',' ');
    nItem=m_ResultsList.InsertItem(nI,str,max(1,m_DQR.GetLevel())-1);
    // Patient ID
    m_ResultsList.SetItem(nItem,m_ColumnPid,LVIF_TEXT,
        dr.GetPatientID(),0,0,0,0);
    // Accession number
    m_ResultsList.SetItem(nItem,m_ColumnAnum,LVIF_TEXT,
        dr.GetAccessionNumber(),0,0,0,0);
    // Modality
    m_ResultsList.SetItem(nItem,m_ColumnMod,LVIF_TEXT/*|LVIF_IMAGE*/,
        dr.GetModality(),0,0,0,0);
    // Study Date
    dr.FormatStudyDate(s,64,false);
    m_ResultsList.SetItem(nItem,m_ColumnSDate,LVIF_TEXT,s,0,0,0,0);
    // Study Time
    dr.FormatStudyTime(s,64,false);
    m_ResultsList.SetItem(nItem,m_ColumnSTime,LVIF_TEXT,s,0,0,0,0);
    // Birth Date
    dr.FormatPatientBirthDate(s,64,false);
    m_ResultsList.SetItem(nItem,m_ColumnBDate,LVIF_TEXT,s,0,0,0,0);
    // Number of study related images
    m_ResultsList.SetItem(nItem,m_ColumnSImgNum,LVIF_TEXT,
        dr.GetStudyImagesNum(),0,0,0,0);
    // Associate array index with item data, to allow for list sorting
    m_ResultsList.SetItemData(nItem,nI);
    // Birth Time - do not use
    //dr.FormatPatientBirthTime(s,64,false);
    //m_ResultsList.SetItem(nItem,m_ColumnBTime,LVIF_TEXT,s,0,0,0,0);
}
// "Higher level" item
nItem=m_ResultsList.InsertItem(0,(const char*)(GetLevelPrompt(true)),4);
m_ResultsList.SetItemData(nItem,m_ParentListItemData);
// Prevent horizontal scroll
m_ResultsList.SetColumnWidth(m_ColumnAnum,LVSCW_AUTOSIZE_USEHEADER);
// Results label
if(nFound>0)
{
    if(nFound==1) res.Format("Results: 1 match found");
    else res.Format("Results: %d matches found",nFound);
    res += res_lev;
}
GetDlgItem(IDC_RESULTS_FRAME)->SetWindowText(res);
GetDlgItem(IDC_STATIC_PROGRESS)->SetWindowText("");
// Size columns to their data
/*
for(int i=0; i<=6; i++)
{
    if(i!=m_ColumnPname && i!=m_ColumnPid) continue;
    m_ResultsList.SetColumnWidth(i,LVSCW_AUTOSIZE_USEHEADER);
}
*/
UpdateData(FALSE);
return true;
}

/*****
*
* Get current selection from the Results list
* and corresponding found object index
* Returns number of parameters successfully retrieved
*
*****/
int DQRControl::GetResultsListSelection(int& sel, int& dcm_index)
{
    sel=dcm_index=-1;
    POSITION pos = m_ResultsList.GetFirstSelectedItemPosition();
    int result;
    if (pos == NULL) result=0; // nothing selected

```

```

else
{
    sel = m_ResultsList.GetNextSelectedItem(pos);    //single selection
    dcm_index=(int)(m_ResultsList.GetItemData(sel));
    if(dcm_index<0 || dcm_index>=m_DQR.GetFoundCount())
    {
        dcm_index=-1;    result=1;
    }
    else
    {
        result=2;
    }
}
m_ButtonGet.EnableWindow(result>1); m_ButtonMoveTo.EnableWindow(result>1);
return result;
}
/*****
*
*   Clicks on Results list
*
*****/
bool DQRControl::OnClickListResults(NMHDR* pNMHDR, LRESULT* pResult)
{
    if(pResult) *pResult = 0;
    int sel, dcm;
    bool item=(GetResultsListSelection(sel, dcm)==2);
    return item;
}

void DQRControl::OnDoubleClickListResults(NMHDR* pNMHDR, LRESULT* pResult)
{
    *pResult = 0;
    int sel, dcm;
    int item=GetResultsListSelection(sel, dcm);
    int nlev=max(1,m_DQR.GetLevel())-1;
    switch (item)
    {
        case 1:// go to higher level
            if(m_DQR.IsOnRootLevel())    OnButtonStartSearch();
            else
            {
                m_RequestedClientService=find_previous;
                if(!AfxBeginThread(StartQRClient,this,GetThreadPriority(),
                    m_ClientStackSize)) RunClientThread();
            }
            return;
        case 2:// go to lower level
            if(m_DQR.IsOnBottomLevel()) { OnButtonGet(); return; }
            else
            {
                m_ResultsListSelection[nlev]=sel;
                m_DOBindex = dcm;
                m_RequestedClientService=find_next;
                if(!AfxBeginThread(StartQRClient,this,GetThreadPriority(),
                    m_ClientStackSize)) RunClientThread();
                return;
            }
    }
    return;
}

void DQRControl::OnRightClickListResults(NMHDR* pNMHDR, LRESULT* pResult)
{
    if(!OnClickListResults(pNMHDR, pResult)) return;
    // Click positioning
    CPoint p(GetMessagePos());
    m_ResultsList.ScreenToClient(&p);

    // Create popup menu
    CMenu menu;
    menu.CreatePopupMenu();
    CMenu menu1;
    menu1.LoadMenu(IDR_MENU_DQRCONTROL);
    CMenu* pop=menu1.GetSubMenu(1);

    // Set acceptable menu IDs

```

```

UINT ids[2]={    IDC_BUTTON_GET,
                 IDC_BUTTON_MOVETO};

// Create the popup menu
int      n=0;
unsigned int  nid;
CString  strMenu, remAET;
for(unsigned int i=0; i<pop->GetMenuItemCount(); i++)
{
    nid=pop->GetMenuItemID(i);
    if( nid!=ids[0] && nid!=ids[1] )    continue;
    if(nid==IDC_BUTTON_GET) m_ButtonGet.GetWindowText(strMenu);
    else if(nid==IDC_BUTTON_MOVETO)
    {
        m_ButtonMoveTo.GetWindowText(strMenu);  strMenu.TrimRight(" :");
        m_MoveArchiveList.GetWindowText(remAET);
        strMenu += CString(" ") + remAET;
    }
    else    pop->GetMenuString(i, strMenu, MF_BYPOSITION);
    int status=pop->GetMenuState(i, MF_BYPOSITION);
    menu.InsertMenu(n, status, nid, strMenu);
    n++;
}

// Display popup menu
ClientToScreen(&p);
p=CPoint(p.x,p.y);
menu.TrackPopupMenu(TPM_LEFTALIGN, p.x,p.y,this);
pop->DestroyMenu();
menu.DestroyMenu();
menu1.DestroyMenu();

/*****
*
* Start the dialog
*
*****/
BOOL DQRControl::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_HWND=GetSafeHwnd();
    LoadArchiveList(m_DQR.GetCurrentAEIndex());
    LoadFoundList(true);
    m_ButtonCancel.ShowWindow(SW_HIDE);
    m_AnimNetwork.ShowWindow(SW_HIDE);
    GetDlgItem(IDC_STATIC_PROGRESS)->ShowWindow(SW_HIDE);
    if(!m_LocalOnly)
    {
        GetDlgItem(IDC_BUTTON_DELETE)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_ARCHIVE_LOCATION)->SetWindowPos(this, 0, 0, 90, 14,
            SWP_NOMOVE | SWP_NOZORDER | SWP_SHOWWINDOW);
    }
    else
        GetDlgItem(IDC_CONNECTION)->ShowWindow(SW_HIDE);

    /* SET m_ResultsList : */
    // 1. Load icons
    if(m_ImageList.m_hImageList==NULL)
    {
        if(!m_ImageList.Create(16,17, ILC_COLOR4,5,1))
        {
            return FALSE;
        }
        m_ImageList.Add(theApp.LoadIcon(IDI_ICON_LEVEL_PATIENT));
        m_ImageList.Add(theApp.LoadIcon(IDI_ICON_LEVEL_STUDY));
        m_ImageList.Add(theApp.LoadIcon(IDI_ICON_LEVEL_SERIES));
        m_ImageList.Add(theApp.LoadIcon(IDI_ICON_LEVEL_IMAGE));
        m_ImageList.Add(theApp.LoadIcon(IDI_ICON_LEVEL_HIGHER));
    }
    m_ResultsList.SetImageList(&m_ImageList, LVSIL_SMALL);
    // 2. Load columns
    m_ResultsList.InsertColumn(m_ColumnPname, "Patient Name", LVCFMT_CENTER, 120, 0);
    m_ResultsList.InsertColumn(m_ColumnPid, "Patient ID", LVCFMT_CENTER, 80, 0);
    m_ResultsList.InsertColumn(m_ColumnBDate, "Birth Date", LVCFMT_CENTER, 80, 0);

```

```

m_ResultsList.InsertColumn(m_ColumnSDate,"Study Date", LVCFMT_CENTER,80,0);
m_ResultsList.InsertColumn(m_ColumnSTime,"Study Time", LVCFMT_CENTER,80,0);
m_ResultsList.InsertColumn(m_ColumnMod,"Modality", LVCFMT_CENTER,60,0);
m_ResultsList.InsertColumn(m_ColumnSImgNum,"Images", LVCFMT_CENTER,50,0);
m_ResultsList.InsertColumn(m_ColumnAnum,"Access. #", LVCFMT_LEFT,100,0);
//m_ResultsList.InsertColumn(m_ColumnBTime,"Birth Time", LVCFMT_CENTER,80,0);
m_ResultsList.SetColumnWidth(m_ColumnAnum,LVSCW_AUTOSIZE_USEHEADER);
// 3. Force entire row selection
m_ResultsList.SetExtendedStyle(LVS_EX_FULLROWSELECT | LVS_EX_SUBITEMIMAGES
    | LVS_EX_GRIDLINES);
// Load all data if required
FindAll();
return TRUE; // return TRUE unless you set the focus to a control
              // EXCEPTION: OCX Property Pages should return FALSE
}

/*****
*
*   Set log files (client and server)
*
*****/
bool DQRControl::CreatedQRControl(DICOMViewLog* ptrClientLog, DICOMDatabase* ptrDB,
    bool local_only)
{
    static bool getIP=true, startServers=true;
    if(!ptrClientLog || !ptrDB)
    {
        AfxMessageBox("Cannot initialize Query/Retrieve control with NULL parameters");
        return false;
    }
    // Set parameters
    if(!m_DQR.CreateQR(ptrClientLog, ptrDB))
    {
        AfxMessageBox("Failed to initialize Query/Retrieve control");
        return false;
    }
    m_DQR.SetDQRCallBack(DQRCtrlCallbackFilter, this);
    m_TaskView.AttachDQR(&m_DQR);
    m_LocalOnly=local_only;
    m_UseTaskQueue = !m_LocalOnly; // Queue tasks on remote only
    m_PromptForTaskScheduler = m_UseTaskQueue; // Prompt for schedule by default
    m_PreloadData = m_LocalOnly; // Preload patient data on local
    m_AEArray = m_DQR.GetAEListPtr();
    // Find local AE at first call
    if(getIP)
    {
        BYTE ip1, ip2, ip3, ip4;
        CString comp_name;
        if(GetLocalIP(comp_name, ip1,ip2,ip3,ip4))
        {
            if(!m_AEArray->SetLocalAE(ip1,ip2,ip3,ip4, (char*)(LPCSTR)comp_name))
            {
                AfxMessageBox("Cannot find Application Entity for this PC");
                return false;
            }
        }
        getIP=false;
    }
    if(!StartTaskQueue())
    {
        AfxMessageBox("Failed to initialize background task queue");
        return false;
    }
    // Start all servers at first call
    if(startServers)
    {
        StartAllServers();
        startServers=false;
    }
    // Retrieve serialized task data
    SerializeDQRControl(true);
    return true;
}

```

```

/*****
*
*   Test network connection to current remote archive (C-ECHO)
*
*****/
void DQRControl::TestArchiveConnection()
{
    Beep(500,100);
    m_RequestedClientService=echo;
    if( !AfxBeginThread(StartQRClient,this,
        GetThreadPriority(),m_ClientStackSize)) RunClientThread();
}

/*****
*
*   Delete. Works with local database only !
*
*****/
void DQRControl::OnButtonDelete()
{
    if(!m_LocalOnly)    return;
    int sel=-1, dcm=-1;
    DICOMRecord dr;
    // Anything selected ?
    if(GetResultsListSelection(sel, dcm) == 2)
    {
        if(!m_DQR.GetFoundRecord(dr,dcm))    return;
    }
    else
    {
        DQRSearch ds;
        if(ds.DoModal() != IDOK)    return;
        ds.WriteIntoDICOMRecord(dr);
    }
    if(AfxMessageBox("Delete specified items from the local database ?",
        MB_YESNO) != IDYES) return;
    int n = m_DQR.DeleteFromLocalDB(dr);
    if(n>0) // Repeat last Find to refresh the worklist window
    {
        m_RequestedClientService=find;
        if( !AfxBeginThread(StartQRClient,this,
            GetThreadPriority(),m_ClientStackSize)) RunClientThread();
    }
}

/*****
*
*   Cancel last network CFind, CGet or CMove
*
*****/
void DQRControl::OnButtonCancel()
{
    Beep(500,100);
    m_DQR.Cancel();
    GetDlgItem(IDC_STATIC_PROGRESS)->SetWindowText("Cancelling requested, wait ...");
}

void DQRControl::EnableCancel()
{
    if(m_RequestedClientService != echo)
    {
        m_ButtonCancel.ShowWindow(SW_SHOW);
    }
}

/*****
*
*   Start search
*
*****/
void DQRControl::OnButtonStartSearch()
{

```

```

long Image::TR_Sobel(const int x, const int y)
{
    long a00=GetLuminance(x-1,y-1); long a10=GetLuminance(x,y-1);
    long a20=GetLuminance(x+1,y-1); long a01=GetLuminance(x-1,y);
    /*long a11=GetLuminance(x,y);*/
    long a21=GetLuminance(x+1,y); long a12=GetLuminance(x,y+1);
    long a02=GetLuminance(x-1,y+1); long a22=GetLuminance(x+1,y+1);

    long tx=(a22-a02)+(a20-a00)+2*(a21-a01);
    long ty=(a00-a02)+(a20-a22)+2*(a10-a12);
    long t=(long)(sqrt(tx*tx+ty*ty));
    if(t<m_EdgeThreshold) t=0;
    if(t>m_maxPixelValue) t=m_maxPixelValue;
    return t;
}

/*****
*
*   Apply Hurst fractal operator
*
*****/
long Image::TR_Fractal(const int x, const int y)
{
    long    p, pmin, pmax;
    double  t, s1=0.0, s2=0.0;

    // Find log variance for each distance from the central pixel
    // d=1
    p=GetLuminance(x-1,y); pmax=pmin=p;
    p=GetLuminance(x+1,y); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x,y-1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x,y+1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    t=log(max(pmax-pmin,1));
    s1 += 0.0; s2 += t;
    // d=sqrt(2)
    p=GetLuminance(x-1,y-1); pmax=pmin=p;
    p=GetLuminance(x+1,y-1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x-1,y+1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x+1,y+1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    t=log(max(pmax-pmin,1));
    s1 += t*0.34657359; s2 += t; //we use ln(sqrt(2));
    // d=2
    p=GetLuminance(x-2,y); pmax=pmin=p;
    p=GetLuminance(x+2,y); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x,y-2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x,y+2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    t=log(max(pmax-pmin,1));
    s1 += t*0.69314718; s2 += t;
    // d=sqrt(5)
    p=GetLuminance(x-1,y-2); pmax=pmin=p;
    p=GetLuminance(x-1,y+2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x+1,y-2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x+1,y+2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x-2,y-1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x-2,y+1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x+2,y-1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x+2,y+1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    t=log(max(pmax-pmin,1));
    s1 += t*0.80471896; s2 += t;
    // d=sqrt(8)
    p=GetLuminance(x-2,y-2); pmax=pmin=p;
    p=GetLuminance(x+2,y-2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x-2,y+2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x+2,y+2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    t=log(max(pmax-pmin,1));
    s1 += t*1.03972077; s2 += t;
    // d=3
    p=GetLuminance(x-3,y); pmax=pmin=p;
    p=GetLuminance(x+3,y); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x,y-3); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x,y+3); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    t=log(max(pmax-pmin,1));
    s1 += t*1.09861229; s2 += t;
}

```

```

// d=sqrt(10)
p=GetLuminance(x-1,y-3);    pmax=pmin=p;
p=GetLuminance(x-1,y+3);    if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
p=GetLuminance(x+1,y-3);    if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
p=GetLuminance(x+1,y+3);    if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
p=GetLuminance(x-3,y-1);    if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
p=GetLuminance(x-3,y+1);    if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
p=GetLuminance(x+3,y-1);    if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
p=GetLuminance(x+3,y+1);    if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
t=log(max(pmax-pmin,1));
s1 += t*1.15129255; s2 += t;
s1 *= 0.14285714;    s2 = s2*0.10477684;

//Find fractal dimension as slope of the Hurst line
//t=max(6.386491206*(s1-s2),0.0); // fractal dimension
t=max(1000*(s1-s2),0.0); // fractal dimension
return( (long)(t) );
}

/*****
*
*   Apply a pixel neighborhood function (with code mask_type) to the image
*
*****/
bool Image::TR_PixelNeighborhood(char mask_type, bool show_progress)
{
    int    rad,i,j,j1,top,bot;
    long    p, pmin, pmax;
    double  tr_scale=1.0;

    // Set pointer to the masking function
    long (Image::*maskF)(const int x, const int y) = 0;
    switch(mask_type)
    {
        case 'g': maskF=TR_DeNoise;rad=2;    break; // denoising
        case 'a': maskF=TR_Smooth;rad=1;    break; // average smoothing
        case 's': maskF=TR_Sharp;rad=1;    break; // sharpening
        case 'e': maskF=TR_Sobel;rad=1;    break; // edge detector
        case 'f': maskF=TR_Fractal;rad=3;    break; // fractal dimension
        default: return false; // invalid mask type
    }

    // Create temporary buffer
    int w = GetWidth();
    int h = GetHeight();
    long *(*buf)=new long*[rad+1];
    if(!buf)
    {
        AfxMessageBox("Low memory, cannot transform");
        return false;
    }
    for(i=0; i<=rad; i++)
    {
        buf[i] = new long[w];
        if(buf[i]==0)
        {
            AfxMessageBox("Low memory, cannot transform");
            for(j=0; j<i; j++) { if(buf[j]) delete [] buf[j]; }
            delete [] buf;
            return false;
        } // out of memory
    }

    // Display progress control in the main frame status bar
    if(show_progress)    theApp.ShowProgress(5,"Transforming ...");

    // Find transform scaling factor
    if(mask_type!='e' && mask_type!='f')
    {
        pmin=0; tr_scale=1.0;
    }
    else
    {

```



```

// Estimate transformed pixel value range
p=pmin=pmax=(this->*maskF)(rad+1,rad+1);
int dw = w>>3;  if(dw<1) dw=1;
int dh = h>>3;  if(dh<1) dh=1;
for(i=rad+1; i<w-rad; i += dw)
{
    for(j=rad+1; j<h-rad; j += dh)
    {
        p=(this->*maskF)(i,j);
        if(p>pmax) pmax=p;
        else if(p<pmin) pmin=p;
    }
    pmax++;
    tr_scale = (double)(m_maxPixelValue)/(pmax-pmin);
}
if(show_progress) theApp.ShowProgress(10);

// Backup pixel values
ResetPixels(true);

// Apply (2*rad-1)*(2*rad-1) masking operator
bot=-1;
top=rad-1;
int procent=0;
for(j=rad; j<=h; j++)
{
    if(show_progress && j%50 == 0)
    {
        procent = 10+(90*j)/h;
        if(procent%3==0) theApp.ShowProgress(procent);
    }
    j1=j-rad-1;
    if(j1>=rad)
    {
        for(i=rad; i<w-rad; i++) SetPixel(i,j1,buf[bot][i]);
    }
    bot = (bot+1)%(rad+1);
    top = (top+1)%(rad+1);
    if(j<h-rad)
    {
        for(i=rad; i<w-rad; i++)
        {
            buf[top][i]=(long)( tr_scale*( (this->*maskF)(i,j)-pmin ) );
        }
    }
}

// Clean up
for(j=0; j<=rad; j++) { if(buf[j]) delete [] buf[j]; }
delete [] buf;
theApp.ShowProgress(0);
Beep(500,50);
return true;
}

```

```

/*****
*
*   Inverts the image
*
*****/
bool Image::TR_Negate()
{
    if(m_pPal->p_active) m_pPal->Negate();
    else

```

```

{
    // Display progress control in the main frame status bar
    theApp.ShowProgress(1,"Changing to negative image ...");
    // Backup pixel values
    ResetPixels(true);
    // Invert pixels values
    if(m_RGB)
    {
        BYTE r, g, b;
        long p;
        for(long i=0; i<(long)m_numPixels; i++)
        {
            if(i%1000==0) theApp.ShowProgress((99*i)/m_numPixels);
            p=GetPixel(i);
            r=m_maxPixelValue-GetRValue(p);
            g=m_maxPixelValue-GetGValue(p);
            b=m_maxPixelValue-GetBValue(p);
            SetPixel(i,RGB(r,g,b));
        }
    }
    else
    {
        for(long i=0; i<(long)m_numPixels; i++)
        {
            if(i%1000==0) theApp.ShowProgress((99*i)/m_numPixels);
            SetPixel(i,m_maxPixelValue-GetPixel(i));
        }
    }
}
theApp.ShowProgress(0);
Beep(500,100);
return true;
}

*****
Copy pixels from current to safe buffer (on true)
or vice versa (on false)
*****/
void Image::ResetPixels(bool current_to_safe)
{
    if(current_to_safe) // current -> safe
    {
        if(m_UndoFile != "+") return;
        m_UndoFileCount++;
        m_UndoFile.Format("%s\\_pix%04d.tmp",theApp.app_DirectoryTmp,m_UndoFileCount);
        FILE* fp = fopen(m_UndoFile,"wb");
        if(!fp) { m_UndoFile = "+"; return; }
        //fwrite(m_Pixels,1,m_numPixelBytes,fp);
        SerializeImage(fp, false);
        fclose(fp);
    }
    else // safe -> current
    {
        if(m_UndoFile == "+" || m_UndoFile == "") return;
        FILE* fp = fopen(m_UndoFile,"rb");
        if(!fp) { return; }
        //fread(m_Pixels,1,m_numPixelBytes,fp);
        SerializeImage(fp, true);
        fclose(fp);
        m_UpdateBitmap=true;
    }
}

/*****
*
* Complete image serialization into a binary file
*
*****/

```

```

bool Image::SerializeImage(FILE *fp, bool is_loading)
{
    int width, height, bpp;
    if(!is_loading)
    {
        width = m_Width;
        height = m_Height;
        bpp = m_Bytes_per_Pixel;
    }

    if(!::SerializeInteger(fp, width, is_loading)) return false;
    if(!::SerializeInteger(fp, height, is_loading)) return false;
    if(!::SerializeInteger(fp, bpp, is_loading)) return false;
    if(is_loading)
    {
        // reformat the image if needed
        if(width!=m_Width || height!=m_Height || bpp!=m_Bytes_per_Pixel)
        {
            if(!CreateImage(width, height, bpp)) return false;
        }
    }
    if(!m_ScreenMap.SerializeScreenMap(fp, is_loading)) return false;
    if(!m_DICOMRecord.SerializeDICOMRecord(fp, is_loading)) return false;
    if(!is_loading) fwrite(m_Pixels,1,m_numPixelBytes,fp);
    else fread(m_Pixels,1,m_numPixelBytes,fp);
    return true;
}

```

```

*****
Histogramm stretch from [amin,amax] to [bmin,bmax] color range.
R,G and B components are stretched together
*****/
bool Image::TR_HistStretch(int amin,int amax,int bmin,int bmax,
                           bool show_progress)

    long i, cmax, p;

    /* Create color map */
    if(m_pPal->p_active) cmax=m_pPal->p_Size;
    else Get_Pixel_minmax(i,cmax);
    cmax++;
    long* color_map;
    try { color_map=new long[cmax]; }
    catch(...)
    {
        AfxMessageBox("Low memory, cannot perform this transform",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
    if(!color_map)
    {
        AfxMessageBox("Low memory, cannot perform this transform",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    } // out of memory

    /* Set color map parameters */
    if(amin<0) amin=0;
    if(bmin<0) bmin=0;
    if(amax>m_maxPixelValue) amax=m_maxPixelValue;
    if(bmax>m_maxPixelValue) bmax=m_maxPixelValue;
    if (amin>=amax || bmin>=bmax) // invalid map
    {
        delete [] color_map; return false;
    }
    if(amin==bmin && amax==bmax) // no stretch needed
    {
        delete [] color_map; return true;
    }

```

```

/* Fill the color map */
long da=amax-amin;
long db=bmax-bmin;
for(i=0; i<cmax; i++)
{
    p=bmin+(db*(i-amin))/da;
    if(p<bmin) p=bmin; else if (p>bmax) p=bmax;
    color_map[i]=p;
}

SetPalette(color_map, cmax, show_progress);

delete [] color_map;
return true;
}

/*****
 *
 * Histogramm stretch from (percent)% median neighborhood
 * to the maximal [0,m_maxPixelValue] range
 *
 *****/
bool Image::TR_HistStretch(BYTE percent, bool show_progress)
{
    long i, amin, amax, amed, p;

    /* Validation */
    if (percent>=100) percent=99;
    if(percent<0) return true;

    /* Find pixel statistics */
    Get_Pixel_minmax(amin,amax);
    if(percent==0) return TR_HistStretch(amin,amax,0,
                                         m_maxPixelValue,show_progress); // simple stretch

    /* Initialize image histogram */
    long cmax=amax+1;
    int* hist;
    try { hist=new int[cmax]; }
    catch(...)
    {
        AfxMessageBox("Low memory, cannot perform this transform",
                      MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
    if(!hist)
    {
        AfxMessageBox("Low memory, cannot perform this transform");
        return false;
    } // out of memory
    for(i=0; i<cmax; i++) hist[i]=0;

    /* Estimate image histogram */
    int hmax=20000;
    int di=__max(1,m_numPixels/10000);
    if(m_RGB) // Color image
    {
        for(i=0; i<m_numPixelBytes; i += di)
        {
            p=m_Pixels[i];
            if(hist[p]<hmax) hist[p] += di;
        }
    }
    else // Greyscale image
    {
        for(i=0; i<(long)m_numPixels; i += di)
        {
            p=GetPixel(i);
            if(hist[p]<hmax) hist[p] += di;
        }
    }
}

```

```

/* Find histogram color average */
double ptot=0, tot=0;
for(i=0; i<cmax; i++)
{
    ptot += ((double)i)*hist[i];
    tot += hist[i];
}
amed = (long)(0.5+ptot/tot);

/* Find new intensity range to preserve */
double keep_max=(100-percent)*tot/100; // number of pixels to keep
double keep=hist[amed];
long bmin=amed; long bmax=amed;
do // do at least once to guarantee bmin<bmax
{
    if(bmin>amin)
    {
        bmin--;
        keep += hist[bmin];
    }
    if(bmax<amax)
    {
        bmax++;
        keep += hist[bmax];
    }
} while (keep<=keep_max);
delete [] hist;

if( ((bmin!=amin)|| (bmax!=amax)) && (bmin<bmax) )
    return TR_HistStretch(bmin,bmax,0,m_maxPixelValue,show_progress);
else return false;

```

\*\*\*\*\*

Histogramm equalization

\*\*\*\*\*

```
bool Image::TR_HistEqualize(bool show_progress)
```

```
    long i, amin, amax, p;
```

```
    /* Find pixel statistics */
    Get_Pixel_minmax(amin,amax);
    Beep(300,100);

```

```
    /* Initialize image histogram, also used as color map */

```

```
    long cmax=amax+1;
    long* hist=new long[cmax];
    if(!hist)
    {

```

```
        AfxMessageBox("Low memory, cannot perform this transform");
        return false;
    } // out of memory

```

```
    for(i=0; i<cmax; i++) hist[i]=0;

```

```
    /* Compute image histogram */

```

```
    long hmax=2000000;
    int di=__max(1,m_numPixels/10000);
    if(m_RGB) // Color image
    {

```

```
        for(i=0; i<m_numPixelBytes; i += di)
        {
            p=m_Pixels[i];
            if(hist[p]<hmax) hist[p] += di;
        }
    }

```

```
    else // Greyscale image
    {

```

```
        for(i=0; i<(long)m_numPixels; i += di)
        {
            p=GetPixel(i);

```

```

        if(hist[p]<hmax) hist[p] += di;
    }

/* Integrate the histogram */
double httotal=0;
for(i=0; i<cmax; i++) httotal += hist[i];
if(httotal<1) // did we have negative pixels or empty image ?
{
    delete [] hist; return false;
}
if(hist[0]<httotal-1) { httotal -= hist[0]; hist[0]=0; }

/* Fill the color map */
double hcum=0;
for(i=0; i<cmax; i++)
{
    hcum += hist[i]; // update cumulative hist
    hist[i]=(long)((m_maxPixelValue*hcum)/httotal);
}

/* Remap the pixel data */
SetPalette(hist, cmax, show_progress);

/* Clean up */
delete [] hist;
return true;
}

*****
Image Gamma correction
*****
bool Image::TR_GammaCorrection(double gamma)
{
    long i, p, pmax;

    /* Validation */
    if(gamma<=0.1) gamma=0.1;
    if(fabs(gamma-1.0)<0.1) return true; // gamma is nearly 1.0, no correction

    /* Find maximum pixel value */
    Get_Pixel_minmax(i,pmax);
    if(pmax==0) return true; //blank image, no correction

    /* Create color map */
    long* color_map;
    try { color_map=new long[pmax+1]; }
    catch(...)
    {
        AfxMessageBox("Low memory, cannot run gamma correction",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
    if(!color_map)
    {
        AfxMessageBox("Low memory, cannot run gamma correction",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    } // out of memory

    /* Fill the color map */
    double c=1.0/pmax;
    double gamma_1=1.0/gamma;
    for(i=1; i<=pmax; i++)
    {
        p=(int)(0.5+pmax*pow(c*i,gamma_1));
        if(p>pmax) color_map[i]=pmax;
        else color_map[i]=p;
    }
}

```

```

    }
    color_map[0]=0;

    // Display progress control in the main frame status bar
    theApp.ShowProgress(1,"Performing gamma correction ...");

    /* Remap the pixel data */
    SetPalette(color_map, pmax+1, true);

    /* Clean up */
    delete [] color_map;
    theApp.ShowProgress(0);
    Beep(500,100);
    return true;
}

/*****
 *
 *   Log transform to enhance dark images
 *
 *****/
bool Image::TR_PixelLog()
{
    long i, p, pmax;

    /* Find maximum pixel value */
    Get_Pixel_minmax(i,pmax);
    if(pmax<=2) return true; //blank image, no correction

    /* Create color map */
    long* color_map;
    try { color_map=new long[pmax+1]; }
    catch(...)
    {
        AfxMessageBox("Low memory, cannot enhance dark image",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
    if(!color_map)
    {
        AfxMessageBox("Low memory, cannot enhance dark image",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    } // out of memory

    /* Fill the color map */
    double c=m_maxPixelValue/log((double)pmax);
    for(i=0; i<=pmax; i++)
    {
        p=(long)( 0.5+c*log((double)(i+1)) );
        if(p>m_maxPixelValue) color_map[i]=m_maxPixelValue;
        else color_map[i]=p;
    }

    /* Remap the pixel data */
    SetPalette(color_map, pmax+1, true);

    /* Clean up */
    delete [] color_map;
    Beep(500,100);
    return true;
}

/*****
 *
 *   Exp transform to enhance light images
 *
 *****/

```

```

*****/
bool Image::TR_PixelExp()
{
    long i, p, pmax;

    /* Find maximum pixel value */
    Get_Pixel_minmax(i,pmax);
    if(pmax<=2) return true; //blank image, no correction

    /* Create color map */
    long* color_map;
    try { color_map=new long[pmax+1]; }
    catch(...)
    {
        AfxMessageBox("Low memory, cannot enhance bright image",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
    if(!color_map)
    {
        AfxMessageBox("Low memory, cannot enhance bright image",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    } // out of memory

    /* Fill the color map */
    double c=1.0/m_maxPixelValue;
    for(i=0; i<=pmax; i++)
    {
        p=(long)( pow((double)m_maxPixelValue,c*i) );
        if(p>m_maxPixelValue) color_map[i]=m_maxPixelValue;
        else color_map[i]=p;
    }

    /* Remap the pixel data */
    SetPalette(color_map, pmax+1, true);

    /* Clean up */
    delete [] color_map;
    Beep(500,100);
    return true;
}

```

```

*****
* Rotate the image
*
*****/
bool Image::TR_Rotate(int degrees)
{
    degrees %= 360;
    if(degrees == 0) return true;
    if(degrees != 90 && degrees != 180 && degrees != 270) return false;
    int x, y, w, h;
    // Do we have 180 degrees ?
    if(degrees == 180)
    {
        // Backup
        ResetPixels(true);
        // Central symmetry
        w = GetWidth(); h = GetHeight();
        long p;
        for(x=0; x<w/2; x++)
        {
            ::ShowProgress((200*x)/w,"Rotating by 180 degrees...");
            for(y=0; y<h; y++)
            {
                p = GetPixel(x,y);
                SetPixel(x,y,GetPixel(w-x,h-y));
                SetPixel(w-x,h-y,p);
            }
        }
    }
}

```



```

    }
    if(w%2==0)
    {
        x = (w/2);
        for(y=0; y<h/2; y++)
        {
            p = GetPixel(x,y);
            'G42Xn          um<m_NumberOfFrames; imagenum++)

{
    info.Format("Loading image # %d/%d",imagenum+1,m_NumberOfFrames);
    // Allocate next image frame
    pImg = &(Add());    if(!pImg)    break;
    if(!(pImg->CreateImage(m_Width,m_Height,(m_BitsAllocated+7)/8,
                           m_Palette)))
    {
        AfxMessageBox("Out of memory for image data", MB_ICONEXCLAMATION|MB_OK);
        RemoveLast();
        pImg = NULL;
        break;
    }
    // Load pixels
    int    percent=0;
    long    pmax=Get(0).m_maxPixelValue; // set acceptable pixel max.
    double    aprog=imagenum*m_Height;
    for(UINT y=0; y<m_Height; y++)
    {
        if(y%60==0)
        {
            percent = (int)((y+aprog)*kprog);
            if(percent%2==0) ::ShowProgress(percent, (char*)(LPCSTR)info);
        }
        for(UINT x=0; x<m_Width; x++)
        {
            p=rpdl.GetBufferedPixel(i);    i++;
            pImg->SetPixelFromLum(x,y,(pmax*(p-p_min))/dp );
        }
    }
    // Set record data
    pImg->SetImageData( &dr, &m_PixelSpacing, &imagenum);
    // Create series palette
    if(imagenum==0) // first image
    {
        m_Palette = new Palette(theApp.app_Metheus,
                                pImg->m_maxPixelValue,
                                pImg->m_RGB);
        pImg->LinkToPalette(m_Palette);
    }
}
vr->Reset();    // Clear data from the vr

// Update image series parameters
if(GetSize()>0)
{
    m_Width=Get(0).GetWidth();
    m_Height=Get(0).GetHeight();
    m_SamplesPerPixel=1;    // we merged multiple samples
    m_BitsAllocated=8*Get(0).GetBytesPerPixel();
    m_BitsStored=m_HighBit=m_BitsAllocated;
}
SetShowSeriesImageInfo(true);
theApp.ShowProgress(0);

return true;
}

/*****
*
* Save bitmap image array as VR data
* This function is required and overrides abstract virtual
* in the base class
*****/

```

```

*
*****/
bool ImageSeries::WritePixels(VR *vr)
{
    int i;
    UINT32 data_size, dsize;
    // Find total data size
    int imagenum=GetUpperBound()+1;
    if(imagenum<=0) return true;
    data_size=0;
    for(i=0; i<imagenum; i++)
    {
        Get(i).GetPixelBytes(dsize);
        data_size += dsize;
    }

    // Allocate pixel encoder
    RawPixelEncoder rpe;
    if(!rpe.SetSize(data_size))
    {
        AfxMessageBox("Out of memory on saving pixel data",
            MB_ICONEXCLAMATION|MB_OK);
        return false;
    }

    // Copy image data into the encoder buffer
    for(i=0; i<imagenum; i++)
    {
        BYTE* data=Get(i).GetPixelBytes(dsize);
        rpe.AddData(data, dsize, Get(i).GetBytesInRow());
    }

    // Attach encoder buffer to the vr
    rpe.TransferDataToVR(vr);

    return true;
}

*****
*
* Get image pointer (safe), and update m_CurrentImageIndex
*
*****/
Image* ImageSeries::GetImage(int n)
{
    // Validate image index
    if(!HasData())
    {
        m_CurrentImageIndex=-1;
        return NULL; // no images
    }
    if(n<0) n=0;
    else if (n>GetUpperBound()) n=GetUpperBound();
    // Retrieve image pointer
    m_CurrentImageIndex=n;
    return &(Get(m_CurrentImageIndex));
}

Image* ImageSeries::GetCurrentImage()
{
    return GetImage(m_CurrentImageIndex);
}

Image* ImageSeries::GetImageFromScreenPoint(CPoint &p)
{
    Image* img = GetCurrentImage();
    if(!img) return NULL;
    if(m_DisplayColumns == 0) return img;
    if(img->ContainsScreenPoint(p)) return img;
    for(int n=0; n<(int)GetSize(); n++)
    {
        if(n==m_CurrentImageIndex) continue;
        if(Get(n).ContainsScreenPoint(p))
        {
            img = GetImage(n);
            SetSelectedImage();
        }
    }
}

```

```

        break;
    }
}
return img;
}

/*****
 *
 *   Sets selection to current image
 *
 *****/
void ImageSeries::SetSelectedImage()
{
    for(unsigned int n=0; n<GetSize(); n++)
    {
        if(Get(n).GetDisplayStatus() != Image::DisplaySelected) continue;
        Get(n).SetDisplayStatus(Image::DisplayNormal);
    }
    Image* img = GetCurrentImage();
    if(img) img->SetDisplayStatus(Image::DisplaySelected);
}

/*****
 *
 *   Adding new DICOMObjects to the series
 *
 *****/
bool ImageSeries::AddDDO(DICOMDataObject &ddo, bool destroy_original)
{
    bool success = true;
    bool read = !HasData();
    ImageSeries tmp_series;
    if(destroy_original) // we can destroy ddo
    {
        if(read) success=ReadDO(ddo);
        else success=tmp_series.ReadDO(ddo);
    }
    else // we must keep the original => use clone
    {
        DICOMDataObject ddo_tmp;
        success = ddo_tmp.CloneFrom(&ddo);
        if(success)
        {
            if(read) success=ReadDO(ddo_tmp);
            else success=tmp_series.ReadDO(ddo_tmp);
        }
    }
    if(!success)
    {
        AfxMessageBox("Cannot add DICOM object to the series",
            MB_ICONEXCLAMATION|MB_OK);
        return false;
    }
    if(read) return success; // no need to append
    else return AddSeries(tmp_series,true);
}

bool ImageSeries::AddDDOFile(CString filename)
{
    DICOMDataObject ddo;
    // Load DDO completely, but without RTC
    if( ddo.LoadFromFile((char*)(LPCSTR)filename, false) )
    {
        return AddDDO(ddo, true);
    }
    else return false;
}

void ImageSeries::AddDICOMRecords(Array<DICOMRecord>& a)
{
    int n=0;
    while (n<(int)a.GetSize())
    {
        if(BelongsToThisSeries(a[n]))
        {

```

```

        AddDDOFile(a[n].GetFileName());
        a.RemoveAt(n);
    }
    else n++;
}
}
/*****
*
*   Adding new series to "this" series, assuming "this" is not empty
*
*****/
bool ImageSeries::AddSeries(ImageSeries &s, bool delete_s)
{
    int ns=s.GetUpperBound();
    if(ns<0) return true; // nothing to do
    int n = GetUpperBound();
    if(n<0) return true; // do not add to empty !

    if(delete_s) Array<Image>::Include(s);
    else SetSize(n+1+ns+1);
    for(int i=0; i<=ns; i++)
    {
        int k=n+i+1;
        if(!delete_s) Get(k).CloneFrom(&s[i]);
        if(k>0)
        {
            Get(k).LinkToPalette(Get(0).m_pPal);
        }
    }

    // Force layout update
    SetDisplayColumns(DisplayColumnsAutomatic);
    SetShowSeriesImageInfo(GetShowSeriesImageInfo());

    return true;
}

/*****
*
*   Test if dr represents an object that belongs to the same series
*   as (this)
*
*****/
bool ImageSeries::BelongsToThisSeries(DICOMRecord &dr)
{
    if(GetSize()<=0) return true;
    return (Get(0).CompareToDICOMRecord(dr, DICOMRecord::LevelSeries) == 0);
}

/*****
*
*   Save the series as a set of BMP files
*   in the given directory
*
*****/
bool ImageSeries::SaveAsImageFiles(CString &directory, CString prefix)
{
    if(!HasData()) return true;
    // Set validated directory and prefix
    directory.TrimLeft(); directory.TrimRight();
    if(!::CreateAndSetCurrentDirectory((char*)(LPCSTR)directory))
    {
        AfxMessageBox("Cannot set image directory\n"+directory,
            MB_ICONEXCLAMATION|MB_OK);
        return false;
    }
    prefix.Replace("\\\\, .?!*_ ", NULL);
    if(prefix=="") prefix="img";
    prefix += "_";
    prefix=directory+CString("\\")+prefix;

    // Store image series
    CString num;
    for(int i=0; i<=GetUpperBound(); i++)

```

```

    {
        num.Format("%s%03d.bmp", prefix, i+1);
        if(!Get(i).WriteToFile(num))
        {
            return false;
        }
    }
    return true;
}

/*****
 *
 *   Set image info for all images in the series
 *
 *****/
void ImageSeries::SetShowSeriesImageInfo(bool show)
{
    for(unsigned int i=0; i<GetSize(); i++) Get(i).SetShowImageInfo(show);
}

bool ImageSeries::GetShowSeriesImageInfo()
{
    if(GetSize()<1) return false;
    return Get(0).GetShowImageInfo();
}

/*****
 *
 *   Display images
 *
 *****/
Image* ImageSeries::DisplayImages(CDC *pDC, CPoint pScroll/*=CPoint(0,0)*/)
{
    // Grab current image pointer
    Image* pImg = GetCurrentImage();
    if(!pImg) return NULL; // no images in this series

    // Initialize screen rectangle, if needed
    if(m_ScreenRect.Width() == 0) SetScreenRect(1.0);

    // Compute image layout, if needed
    if(m_DisplayColumnsChanged)
    {
        SetOptimalImageLayout();
        m_DisplayColumnsChanged=false;
    }

    // Display
    if(m_DisplayColumns == 0) // Single image
    {
        pImg->DisplayDIB(pDC, pScroll);
    }
    else
    {
        for(unsigned int n=0; n<GetSize(); n++)
        {
            Get(n).DisplayDIB(pDC, pScroll);
        }
    }
    return pImg;
}

/*****
 *
 *   Set the number of display columns.
 *
 *****/
bool ImageSeries::SetDisplayColumns(int ncol)
{
    m_DisplayColumnsChanged = (ncol != m_DisplayColumns);
    m_DisplayColumns = ncol;
    return m_DisplayColumnsChanged;
}

/*****

```

```

}
*
* Find the optimal multi-image layout.
*
*****/
void ImageSeries::SetOptimalImageLayout()
{
    int icount=0, n;
    // Process individual image display
    if(m_DisplayColumns == 0)
    {
        for(unsigned n=0; n<GetSize(); n++)
        {
            Get(n).m_ScreenMap.FitInsideScreenRect(m_ScreenRect);
        }
        return;
    }
    // Process multiple image display
    for(n=0; n<(int)GetSize(); n++)
    {
        if(Get(n).GetDisplayStatus() == Image::DisplayHidden) continue;
        icount ++;
    }
    if(icount<1) return; // nothing to display
    int nr, best_nc;
    Image* pImg = GetCurrentImage();
    if(!pImg) return;
    int iw = pImg->GetWidth();
    int ih = pImg->GetHeight();
    const static int d = 16;

    // Find the best image zooming factor
    double best_zoom;
    if(m_DisplayColumns>=0) // Column number was specified
    {
        best_nc = max(1,min(m_DisplayColumns,icount));
        nr = (icount+best_nc-1)/best_nc;
        best_zoom = min( (m_ScreenRect.Width()-d*(best_nc+1.0))/(iw*best_nc),
                        (m_ScreenRect.Height()-d*(nr+1.0))/(ih*nr) );
    }
    else // Column number unknown, find best
    {
        double zoom;
        best_zoom = 0.0;
        for(int nc=1; nc <= 1+icount/2; nc++)
        {
            nr = (icount+nc-1)/nc;
            zoom = min( (m_ScreenRect.Width()-d*(nc+1.0))/(iw*nc),
                        (m_ScreenRect.Height()-d*(nr+1.0))/(ih*nr) );
            if(zoom>best_zoom)
            {
                best_zoom = zoom; best_nc = nc;
            }
        }
        m_DisplayColumns = best_nc;
    }

    // Find image screen sizes
    nr = (icount+best_nc-1)/best_nc;
    int w = max(4, (int)(iw*best_zoom) );
    int h = max(4, (int)(ih*best_zoom) );
    int dx = max(1, (m_ScreenRect.Width()-w*best_nc)/(best_nc+1) );
    int dy = max(1, (m_ScreenRect.Height()-h*nr)/(nr+1) );

    // Set screen viewing rectangles for images
    int x, y;
    CRect r;
    for(n=0; n<(int)GetSize(); n++)
    {
        if(Get(n).GetDisplayStatus() == Image::DisplayHidden) continue;
        x = dx+(n % best_nc)*(w+dx);
        y = dy+(n / best_nc)*(h+dy);
        r.SetRect(x,y,x+w,y+h);
        Get(n).m_ScreenMap.FitInsideScreenRect(r);
    }
}

```

```

    return;
}

/*****
 *
 *   Alternate between browse and non-browse states
 *
 *****/
void ImageSeries::SwitchBrowseView()
{
    if(m_DisplayColumns == 0)    SetDisplayColumns(DisplayColumnsAutomatic);
    else                        SetDisplayColumns(0);
}

/*****
 *
 *   Set screen rectangle from a zoom factor
 *
 *****/
void ImageSeries::SetScreenRect(double zoom/*=1.0*/)
{
    if(zoom<0.1 || zoom > 10.0) return;
    // Find screen size
    ((CMDIFrameWnd*)AfxGetMainWnd())->MDIGetActive()->GetClientRect(m_ScreenRect);
    m_ScreenRect.BottomRight().y -= 20;
    m_ScreenRect.BottomRight().x -= 20;
    if(zoom!=1.0)    // zoom
    {
        int w = max(64, (int)(zoom*m_ScreenRect.Width()));
        int h = max(64, (int)(zoom*m_ScreenRect.Height()));
        int x = max(0, (m_ScreenRect.Width()-w)/2);
        int y = max(0, (m_ScreenRect.Height()-h)/2);
        m_ScreenRect.SetRect(x,y,x+w,y+h);
    }
    SetOptimalImageLayout();
}

/*****
 *
 *   Zoom and offset images
 *
 *****/
void ImageSeries::ZoomImages(double zoom)
{
    for(unsigned int n=0; n<GetSize(); n++)
    {
        Get(n).SetImageRectZoom(zoom);
    }
}

void ImageSeries::OffsetImages(double dx, double dy)
{
    for(unsigned int n=0; n<GetSize(); n++)
    {
        Get(n).SetImageRectOffset(dx, dy);
    }
}

double ImageSeries::GetZoom()
{
    Image* pImg = GetCurrentImage();
    if(pImg)    return pImg->GetZoom();
    else        return 1.0; // no zoom
}

```

```
// Palette.h: interface for the Palette class.
//
///////////////////////////////////////////////////////////////////

#ifndef AFX_PALETTE_H_INCLUDED_
#define AFX_PALETTE_H_INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class Palette
{
public:
    bool        p_active;
    int         p_Size;

    void        BackUp();
    void        Negate(bool RGB=false);
    void        Get_Pal_minmax(long& pmin, long& pmax, bool RGB=false);
    bool        CreateNewPalette(bool Metheus, long max_color, bool rgb);
    bool        CloneFromPalette(Palette *pPal);
    bool        IsCorrectPalette();
    bool        LoadPalette(CDC* pDC, bool RGB=false);
    bool        SetPalette();
    bool        SetPalette(long* color_map, long color_map_size);
    bool        SetPalette(long offset, double stretch);
    inline long GetPaletteColor(long index);

    Palette();
    Palette(bool Metheus, long max_color, bool rgb);
    ~Palette();

private:
    bool        p_Metheus;
    bool        p_update;
    int         p_factor;
    long        p_maxCol;
    USHORT*     p_Val;
    USHORT*     p_Val_backup;
    HPALETTE    p_Hpal;

    void        DeletePalette();
};

/*
 *
 * Return palette color at the given index
 *
 */
///////////////////////////////////////////////////////////////////
inline long Palette::GetPaletteColor(long index)
{
    if(index<0)            index=0;
    else if(index>=p_Size) index=p_Size-1;
    return p_Val[index];
}

#endif // !defined(AFX_PALETTE_H_INCLUDED_)
```



1

```

    p_Val_backup=new USHORT[p_Size];
    if(p_Val_backup)    p_Val=new USHORT[p_Size];
    p_active = (p_Val!=NULL);
}
catch(...)
{
    AfxMessageBox("Low memory, cannot allocate image palette",
        MB_OK|MB_ICONEXCLAMATION);
    p_active=false;
    return false;
}
if(p_active)
{
    for(USHORT i=0; i<p_Size; i++) p_Val[i]=i;
    BackUp();
}
p_update=true;

// Only greyscale palettes
if(p_active)    p_active=!rgb;
return true;
}

/*****
*
*   Copying palettes
*
*****/
bool Palette::CloneFromPalette(Palette *pPal)
{
    if(!pPal)    return false;
    if( !CreateNewPalette(pPal->p_Metheus, pPal->p_maxCol,
        !(pPal->p_active)) )    return false;
    if(pPal->p_Val)
    {
        memcpy(p_Val, pPal->p_Val, p_Size*(sizeof USHORT));
        BackUp();
    }
    return true;
}

/*****
*
*   Set identity palette
*
*****/
bool Palette::SetPalette()
{
    if(!p_active || !p_Val) return false;
    for(USHORT i=0; i<p_Size; i++) p_Val[i]=i;
    p_update=true;
    return true;
}

/*****
*
*   General (from array) palette update
*
*****/
bool Palette::SetPalette(long *color_map, long color_map_size)
{
    if(!p_active || !p_Val) return false;

    long    i;
    long    imax= ( p_Size>color_map_size ? color_map_size : p_Size ) ;
    for(i=0; i<imax; i++)    p_Val[i]=(USHORT)labs(color_map[i]);
    for(i=imax; i<p_Size; i++)    p_Val[i]=p_Val[imax-1];
    p_update=true;
    return true;
}

/*****
*
*   Linear palette update for Fast Color/Contrast
*****/

```

```

*
*****
bool Palette::SetPalette(long offset, double stretch)
{
    if(!p_active || !p_Val) return false;

    long i, n;
    long l_offset=offset<<10;
    long l_stretch = (long)(1024*stretch);
    for(i=0; i<p_Size; i++)
    {
        n=(l_offset+((long)p_Val_backup[i])*l_stretch);
        if(n<=0) p_Val[i]=0;
        else
        {
            n >>= 10;
            if(n>=p_maxCol) n=p_maxCol-1;
            p_Val[i]=(USHORT)(n);
        }
    }
    p_update=true;
    return true;
}

/*****
*
* Create and load palette into the given DC
*
*****
bool Palette::LoadPalette(CDC *pDC, bool RGB)
{
    if(!p_Val) return false; // cannot allocate palette
    if(!p_active) return true; // disabled palettes
    long i;
    if(!p_Metheus) // load Windows palette
    {
        if(!p_update) // same palette as before, skip palette reload
        {
            ::SelectPalette(pDC->m_hDC,p_Hpal,TRUE);
            pDC->RealizePalette();
            p_update=false;
            return true;
        }
        BYTE r,g,b;
        int cPalette = sizeof(LOGPALETTE)+sizeof(PALETTEENTRY)*p_Size;
        LOGPALETTE* pPal = (LOGPALETTE*)new BYTE[cPalette];
        if(!pPal) return false;
        pPal->palVersion = 0x300;
        pPal->palNumEntries = (unsigned short)p_Size;
        for(i = 0; i<p_Size; i++)
        {
            if(RGB)
            {
                r=GetRValue(p_Val[i]);
                g=GetGValue(p_Val[i]);
                b=GetBValue(p_Val[i]);
            }
            else r=g=b=p_factor*(BYTE)__min(p_Val[i],p_Size-1);
            pPal->palPalEntry[i].peRed = r;
            pPal->palPalEntry[i].peGreen = g;
            pPal->palPalEntry[i].peBlue = b;
            pPal->palPalEntry[i].peFlags = NULL;
        }
        DeleteObject(p_Hpal); // free display memory
        p_Hpal = CreatePalette(pPal);
        delete [] (BYTE*)pPal;
        if(!p_Hpal) return false;
        ::SelectPalette(pDC->m_hDC,p_Hpal,FALSE);
        pDC->RealizePalette();
        DeleteObject(p_Hpal); // free display memory
    }
    else // Metheus device

```

```

{
    if(!p_update) return true; //same palette as before
    USHORT* p_Val_tmp = new USHORT[p_Size];
    if(!p_Val_tmp) return false;
    for(i=0; i<p_Size; i++) p_Val_tmp[i]=p_factor*p_Val[i];
    if(MetheusLoadGrayPalette(pDC->GetSafeHdc(),
        theApp.app_DynamicPaletteStart,p_Size,p_Val_tmp)==FALSE)
    {
        delete [] p_Val_tmp;
        return false;
    }
    delete [] p_Val_tmp;
}
p_update=false;
return true;
}

/*****
*
* Find min and max palette colors
*
*****/
void Palette::Get_Pal_minmax(long &pmin, long &pmax, bool RGB)
{
    long i, p;
    pmin=0; pmax=1;
    if(!p_Val) return;
    if(!RGB) // greyscale
    {
        pmin=p_Val[0]; pmax=pmin+1;
        for(i=1; i<p_Size; i++)
        {
            if(p_Val[i]>pmax) pmax=p_Val[i];
            else if(p_Val[i]<pmin) pmin=p_Val[i];
        }
    }
    else
    {
        BYTE r,g,b;
        pmin=GetRValue(p_Val[0]); pmax=pmin+1;
        for(i=0; i<p_Size; i++)
        {
            p=p_Val[i];
            r=GetRValue(p); g=GetGValue(p); b=GetBValue(p);
            if(r>pmax) pmax=r;
            else if(r<pmin) pmin=r;
            if(g>pmax) pmax=g;
            else if(g<pmin) pmin=g;
            if(b>pmax) pmax=b;
            else if(b<pmin) pmin=b;
        }
    }
}

/*****
*
* Invert palette colors
*
*****/
void Palette::Negate(bool RGB)
{
    long i, p;
    if(!p_active || !p_Val) return;
    if(!RGB)
    {
        for(i=0; i<p_Size; i++)
        {
            p=p_maxCol-1-(long)p_Val[i];
            if(p<0) p=0; else if(p>=p_maxCol) p=p_maxCol-1;
            p_Val[i]=(USHORT)p;
        }
    }
}

```

```

    }
    else
    {
        BYTE r,g,b;
        for(i=0; i<p_Size; i++)
        {
            r=GetRValue(p_Val[i]); g=GetGValue(p_Val[i]); b=GetBValue(p_Val[i]);
            p_Val[i]=(USHORT)RGB(255-r,255-g,255-b);
        }
        p_update=true;
    }

/*****
*
*   Return "true" if palette supports the same color range as the image
*
*****/
bool Palette::IsCorrectPalette()
{
    return (p_factor==1 || p_Metheus);
}

/*****
*
*   Save current palette colors
*
*****/
void Palette::BackUp()
{
    if(!p_active || !p_Val || !p_Val_backup || !p_Size) return;
    memcpy(p_Val_backup, p_Val, p_Size*(sizeof USHORT));
}

```

```
// ScreenMap.h: interface for the ScreenMap class.
//
/////////////////////////////////////////////////////////////////

#if !defined(AFX_SCREENMAP_H_C4AA3744_77E9_11D2_9586_00105A21774F__INCLUDED_)
#define AFX_SCREENMAP_H_C4AA3744_77E9_11D2_9586_00105A21774F__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class ScreenMap
{
public:
    CRect    crScreen;
    CRect    crImage;

    void      FitInsideScreenRect(CRect screen);
    void      ValidateZoom(double& x);
    void      Rotate(int degrees);
    void      SetLeftTopScreenPoint(int x, int y);
    bool      Initialize(const CRect cr);
    bool      Screen_in_Image(const CPoint screenP, const int bound=0);
    bool      Screen_in_Image(const CRect screenR, const int bound=0);
    bool      SerializeScreenMap(FILE* fp, bool is_loading);
    double    GetZoom();
    CSize     GetScreenCenteredSize();
    CPoint    Image_to_Screen(const CPoint cpI);
    CPoint    Image_to_Screen(const CPoint cpS, const CPoint offset);
    CPoint    Screen_to_Image(const CPoint cpS);
    CPoint    Screen_to_Image(const CPoint cpS, const CPoint offset);
    CRect     Image_to_Screen(const CRect crI);
    CRect     Image_to_Screen(const CRect crI, const CPoint offset);
    CRect     Screen_to_Image(const CRect crS);
    CRect     Screen_to_Image(const CRect crS, const CPoint offset);

    ScreenMap();
    ScreenMap(ScreenMap& sm);
    ~ScreenMap();

private:
};

#endif // !defined(AFX_SCREENMAP_H_C4AA3744_77E9_11D2_9586_00105A21774F__INCLUDED_)
```

```

// ScreenMap.cpp: implementation of the ScreenMap class.
//
//
//
#include "stdafx.h"
#include "..\DCM.h"
#include "ScreenMap.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//
// Construction/Destruction
//
ScreenMap::ScreenMap()
{
}

ScreenMap::ScreenMap(ScreenMap& sm)
{
    crImage = sm.crImage;
    crScreen = sm.crScreen;
}

ScreenMap::~ScreenMap()
{
}

bool ScreenMap::Initialize(const CRect cr)
{
    crImage=cr;
    crScreen=cr;
    return true;
}

CPoint ScreenMap::Screen_to_Image(const CPoint cpS)
{
    CPoint p;
    p = cpS-crScreen.TopLeft();
    p.x= (p.x*crImage.Width())/crScreen.Width();
    p.y= (p.y*crImage.Height())/crScreen.Height();
    return (p+crImage.TopLeft());
}

CPoint ScreenMap::Screen_to_Image(const CPoint cpS, const CPoint offset)
{
    return Screen_to_Image(cpS-offset);
}

CRect ScreenMap::Screen_to_Image(const CRect crS)
{
    return CRect(this->Screen_to_Image(crS.TopLeft()),
        this->Screen_to_Image(crS.BottomRight()) );
}

CRect ScreenMap::Screen_to_Image(const CRect crS, const CPoint offset)
{
    CRect r=crS;
    r.OffsetRect(offset);
    return Screen_to_Image(r);
}

CPoint ScreenMap::Image_to_Screen(const CPoint cpI)
{
    CPoint p;
    p = cpI-crImage.TopLeft();
    p.x= (p.x*crScreen.Width())/crImage.Width();
    p.y= (p.y*crScreen.Height())/crImage.Height();

```

```

    return (p+crScreen.TopLeft());
}
CPoint ScreenMap::Image_to_Screen(const CPoint cpS, const CPoint offset)
{
    return Image_to_Screen(cpS)+offset;
}

CRect ScreenMap::Image_to_Screen(const CRect crI)
{
    return CRect(this->Image_to_Screen(crI.TopLeft()),
        this->Image_to_Screen(crI.BottomRight()) );
}

CRect ScreenMap::Image_to_Screen(const CRect crS, const CPoint offset)
{
    CRect r=Image_to_Screen(crS);
    r.OffsetRect(offset);
    return r;
}

void ScreenMap::ValidateZoom(double & x)
{
    if(x*crImage.Width()<16) x=16.0/crImage.Width();
    else if(x*crImage.Width()>4096) x=4096.0/crImage.Width();
    if(x*crImage.Height()<16) x=16.0/crImage.Height();
    else if(x*crImage.Height()>4096) x=4096.0/crImage.Height();
    int w_screen=4 * ( (2+(int)(0.5*x*crImage.Width()))/4 );
    int h_screen=4 * ( (2+(int)(0.5*x*crImage.Height()))/4 );
    crScreen.SetRect( crScreen.TopLeft().x, crScreen.TopLeft().y,
        crScreen.TopLeft().x+w_screen,
        crScreen.TopLeft().y+h_screen );
}

CSize ScreenMap::GetScreenCenteredSize()
{
    return CSize(crScreen.right+crScreen.left,crScreen.top+crScreen.bottom);
}

void ScreenMap::SetLeftTopScreenPoint(int x, int y)
{
    if(x<0) x=0;
    if(y<0) y=0;
    CSize offset=CPoint(x,y)-crScreen.TopLeft();
    crScreen.OffsetRect(offset);
}

bool ScreenMap::Screen_in_Image(const CPoint screenP, const int bound)
{
    CPoint p=Screen_to_Image(screenP);
    CRect ir=crImage; ir.DeflateRect(bound,bound);
    return (ir.PtInRect(p)==TRUE);
}

bool ScreenMap::Screen_in_Image(const CRect screenR, const int bound)
{
    return (Screen_in_Image(screenR.TopLeft(),bound) &&
        Screen_in_Image(screenR.BottomRight(),bound));
}

/*****
*
* Rotate screen map rectangles by 90 degrees.
*
*****/
void ScreenMap::Rotate(int degrees)
{
    if(degrees != 90 && degrees != 270) return; // nothing to do
    crScreen.SetRect(crScreen.left, crScreen.top,
        crScreen.left+crScreen.Height(),
        crScreen.top+crScreen.Width() );

    crImage.SetRect(crImage.left, crImage.top,
        crImage.left+crImage.Height(),
        crImage.top+crImage.Width() );
}

```



```

/*****
*
*   Serialize this screen map.
*   Used as a part of image serialization
*
*****/
bool ScreenMap::SerializeScreenMap(FILE *fp, bool is_loading)
{
    int ix0,ix1,iy0,iy1,sx0,sx1,sy0,sy1;
    if(!is_loading)
    {
        ix0 = crImage.left;      ix1 = crImage.right;
        iy0 = crImage.top;       iy1 = crImage.bottom;
        sx0 = crScreen.left;     sx1 = crScreen.right;
        sy0 = crScreen.top;      sy1 = crScreen.bottom;
    }
    if(!::SerializeInteger(fp, ix0, is_loading)) return false;
    if(!::SerializeInteger(fp, ix1, is_loading)) return false;
    if(!::SerializeInteger(fp, iy0, is_loading)) return false;
    if(!::SerializeInteger(fp, iy1, is_loading)) return false;
    if(!::SerializeInteger(fp, sx0, is_loading)) return false;
    if(!::SerializeInteger(fp, sx1, is_loading)) return false;
    if(!::SerializeInteger(fp, sy0, is_loading)) return false;
    if(!::SerializeInteger(fp, sy1, is_loading)) return false;
    if(!::SerializeInteger(fp, ix1, is_loading)) return false;
    if(!::SerializeInteger(fp, iy0, is_loading)) return false;

    if(is_loading)
    {
        crImage.SetRect (ix0,iy0,ix1,iy1);
        crScreen.SetRect(sx0,sy0,sx1,sy1);
    }
    return true;
}

/*****
*
*   Place "crScreen" inside given "screen"
*
*****/
void ScreenMap::FitInsideScreenRect(CRect screen)
{
    double c = min(screen.Width()/(1.0+crScreen.Width()),
                    screen.Height()/(1.0+crScreen.Height()));
    int w = (int)max(4,c*crScreen.Width());
    int h = (int)max(4,c*crScreen.Height());
    int x = (screen.TopLeft().x+screen.BottomRight().x)/2;
    int y = (screen.TopLeft().y+screen.BottomRight().y)/2;
    crScreen = CRect(x-w/2,y-h/2,x+w/2,y+h/2);
}

/*****
*
*   Get current map zoom factor
*
*****/
double ScreenMap::GetZoom()
{
    return (crScreen.Width()+0.0001)/(crImage.Width()+0.0001);
}

```

```

#if !defined(AFX_AEOPTIONS_DIALOG_H__INCLUDED_)
#define AFX_AEOPTIONS_DIALOG_H__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// AEOptions_Dialog.h : header file
//

////////////////////////////////////
// AEOptions_Dialog dialog

class AEOptions_Dialog : public CDialog
{
// Construction
public:
    virtual int DoModal(ApplicationEntityList *AEArray);
    AEOptions_Dialog(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{AFX_DATA(AEOptions_Dialog)
    enum { IDD = IDD_DIALOG_AE_OPTIONS };
    BOOL m_useMoveToRetrieve;
    int m_Port;
    int m_PortServer;
    int m_Timeout;
    CString m_Comments;
    CIPAddressCtrl m_IP;
    CComboBox m_AEComboList;
    CString m_Title;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(AEOptions_Dialog)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(AEOptions_Dialog)
    virtual BOOL OnInitDialog();
    virtual void OnOK();
    afx_msg void OnCloseupComboAeList();
    afx_msg void OnAENew();
    afx_msg void OnAEDelete();
    afx_msg void OnAeClone();
    afx_msg void OnSelchangeComboAeList();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()

private:
    int m_ListIndex;
    ApplicationEntityList *m_AEArray;

    void ResetAeList(int new_selection=0);
    void UpdateAllFields(bool from_data_to_dialog=true);
};

//{AFX_INSERT_LOCATION}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_AEOPTIONS_DIALOG_H__INCLUDED_)

```

```
// AEOptions_Dialog.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "..\Resource.h"
#include "AEOptions_Dialog.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// AEOptions_Dialog dialog
```

```
AEOptions_Dialog::AEOptions_Dialog(CWnd* pParent /*=NULL*/)
: CDialog(AEOptions_Dialog::IDD, pParent)
```

```
{
    //{{AFX_DATA_INIT(AEOptions_Dialog)
    m_useMoveToRetrieve = FALSE;
    m_Port = 0;
    m_PortServer = 0;
    m_Timeout = 0;
    m_Comments = T("");
    m_Title = T("");
    //}}AFX_DATA_INIT
    m_ListIndex=0;
```

```
void AEOptions_Dialog::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(AEOptions_Dialog)
    DDX_Check(pDX, IDC_CHECK_MOVETOGET, m_useMoveToRetrieve);
    DDX_Text(pDX, IDC_EDIT_PORT, m_Port);
    DDV_MinMaxInt(pDX, m_Port, 0, 70000);
    DDX_Text(pDX, IDC_EDIT_PORT_SERVER, m_PortServer);
    DDV_MinMaxInt(pDX, m_PortServer, 1, 70000);
    DDX_Text(pDX, IDC_EDIT_TIMEOUT, m_Timeout);
    DDV_MinMaxInt(pDX, m_Timeout, 0, 100000);
    DDX_Text(pDX, IDC_EDIT_COMMENTS, m_Comments);
    DDV_MaxChars(pDX, m_Comments, 63);
    DDX_Control(pDX, IDC_AE_IPADDRESS, m_IP);
    DDX_Control(pDX, IDC_COMBO_AE_LIST, m_AEComboList);
    DDX_Text(pDX, IDC_EDIT_TITLE, m_Title);
    DDV_MaxChars(pDX, m_Title, 16);
    //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(AEOptions_Dialog, CDialog)
```

```
    //{{AFX_MSG_MAP(AEOptions_Dialog)
    ON_CBN_CLOSEUP(IDC_COMBO_AE_LIST, OnCloseupComboAeList)
    ON_BN_CLICKED(ID_AE_NEW, OnAENew)
    ON_BN_CLICKED(ID_AE_DELETE, OnAEDelete)
    ON_BN_CLICKED(ID_AE_CLONE, OnAeClone)
    ON_CBN_SELCHANGE(IDC_COMBO_AE_LIST, OnSelchangeComboAeList)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// AEOptions_Dialog message handlers
```

```
*****
```

```
* Display modal dialog for AE setup
```

```
*****/
```

```
int AEOptions_Dialog::DoModal(ApplicationEntityList *AEarray)
```

```
{
    if(!AEarray) return -1;
    m_AEarray = AEarray;
```

```

    m_ListIndex=m_AEarray->GetCurrentIndex();
    CDialog::DoModal();
    return m_ListIndex;
}

```

```

/*****

```

```

*   Initialize all parameter fields

```

```

*****/

```

```

BOOL AEOptions_Dialog::OnInitDialog()

```

```

{
    CDialog::OnInitDialog();
    ResetAEList(m_ListIndex);
    GotoDlgCtrl(GetDlgItem(IDCANCEL));
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

```

```

/*****

```

```

*   Update all parameter fields

```

```

*****/

```

```

void AEOptions_Dialog::UpdateAllFields(bool from_data_to_dialog)

```

```

{
    int ind = m_AEComboList.GetCurSel();
    if(from_data_to_dialog)
    {
        if(ind>=(int)(m_AEarray->GetSize()) || ind<0) ind=m_ListIndex;
        else m_ListIndex=ind;
        ApplicationEntity* a = &(m_AEarray->Get(ind));
        m_IP.SetAddress(a->ae_IP1,a->ae_IP2,a->ae_IP3,a->ae_IP4);
        m_Title=CString(a->ae_Title);
        m_Port=a->ae_Port; m_PortServer=a->ae_PortServer;
        m_Timeout=a->ae_Timeout;
        m_Comments=CString(a->ae_Comments);
        m_useMoveToRetrieve=a->ae_useMoveAsGet;
        UpdateData(FALSE);
    }
    else
    {
        UpdateData(TRUE);
        if(ind>=(int)(m_AEarray->GetSize()) || ind<0) ind=m_ListIndex;
        else m_ListIndex=ind;
        CString location; m_AEComboList.GetLBText(ind,location);
        BYTE ip1, ip2, ip3, ip4; m_IP.GetAddress(ip1, ip2, ip3, ip4);
        m_AEarray->Get(ind).SetApplicationEntity ((char*)(LPCSTR)m_Title,
                                                    ip1, ip2, ip3, ip4,m_Port,
                                                    m_PortServer,m_Timeout,(char*)(LPCSTR)location,
                                                    (char*)(LPCSTR)m_Comments,m_useMoveToRetrieve==TRUE);
    }
}

```

```

/*****

```

```

*   Combo List message handler

```

```

*****/

```

```

void AEOptions_Dialog::OnCloseupComboAeList()

```

```

{
    int n;
    int ind = m_AEComboList.GetCurSel();
    CString info;
    m_AEComboList.GetWindowText(info);
    if(ind != CB_ERR)
    {
        // If a valid choice was made from a listbox,
        // update all AE parameters
        m_ListIndex=ind;
        UpdateAllFields(true);
    }
    else if(info.IsEmpty() == TRUE)
    {

```



```
// MainFrm.cpp : implementation of the CMainFrame class
//
```

```
#include "stdafx.h"
#include "DCM.h"
#include "MainFrm.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CMainFrame
```

```
IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)
```

```
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    ON_COMMAND(ID_VIEW_TOOLBAR, OnViewToolBar)
    ON_UPDATE_COMMAND_UI(ID_VIEW_TOOLBAR, OnUpdateViewToolBar)
    ON_WM_DROPFILES()
    //}}AFX_MSG_MAP
    ON_UPDATE_COMMAND_UI(ID_PROGRESS_STATUS, OnUpdateProgressStatus)
    // Support dropdown toolbar buttons
    ON_NOTIFY(TBN_DROPDOWN, AFX_IDW_TOOLBAR, OnToolBarDropDown)
    // Global help commands
    ON_COMMAND(ID_HELP_FINDER, CMDIFrameWnd::OnHelpFinder)
    ON_COMMAND(ID_HELP, CMDIFrameWnd::OnHelp)
    ON_COMMAND(ID_CONTEXT_HELP, CMDIFrameWnd::OnContextHelp)
    ON_COMMAND(ID_DEFAULT_HELP, CMDIFrameWnd::OnHelpFinder)
END_MESSAGE_MAP()
```

```
static UINT indicators[] =
```

```
{
    ID_SEPARATOR,           // status line indicator
    ID_PROGRESS_STATUS,
    ID_INDICATOR_CAPS,
    //ID_INDICATOR_NUM,
    //ID_INDICATOR_SCRL,
    ID_DICTIONARY_STATUS
}
```

```
////////////////////////////////////
// CMainFrame construction/destruction
```

```
CMainFrame::CMainFrame()
```

```
{
    m_showToolbars=true;
    /* Dummy string to size progress indicator in the status bar */
    m_paneString=CString(' ',60);
}
```

```
CMainFrame::~CMainFrame()
```

```
{
}
```

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
```

```
{
    if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)    return -1;
    ShowWindow(SW_MAXIMIZE);

    // Create application toolbars
    if (!m_ToolBarBasic.CreateIE(this,32,32,
        IDR_TOOLBAR_BASIC))
    {
        return -1;        // fail to create
    }
    m_ToolBarBasic.AttachDropDown(ID_BUTTON_ROI, IDR_DCMTYPE, ID_BUTTON_SELECT_RECT);
    m_ToolBarBasic.AttachDropDown(ID_VIEW_FLIP, IDR_DCMTYPE, ID_VIEW_FLIP_VERTICAL);
    m_ToolBarBasic.AttachDropDown(ID_BUTTON_MEASURE, IDR_DCMTYPE,
        ID_BUTTON_MEASURE_RULER);
}
```

```

if (!m_ToolBarMultiframe.CreateIE(this,32,32,
    IDR_TOOLBAR_MULTIFRAME))
{
    return -1;        // fail to create
}
m_ToolBarMultiframe.AttachDropDown(ID_BROWSE_FRAME,IDR_DCMTYPE,
    ID_FRAMES_LAYOUT_STACK);
// Create application dialog bar
if (!m_wndDlgBar.Create(this, IDD_BAR_INFO,
    CBRS_ALIGN_RIGHT, AFX_IDW_DIALOGBAR))
{
    TRACE0("Failed to create dialogbar\n");
    return -1;        // fail to create
}

// Create animated logo
if (!m_Animate.Create(WS_CHILD | WS_VISIBLE | ACS_AUTOPLAY,
    CRect(0,0,80,60), this, 0) ||
    !m_Animate.Open(IDR_AVI_DCM))
{
    TRACE0("Failed to create animation control\n");
    return -1;        // fail to create
}

// Create application ReBar
if (!m_ReBar.Create(this,0) ||
    !m_ReBar.AddBar(&m_Animate, NULL, NULL,RBBS_FIXEDBMP | RBBS_FIXEDSIZE) ||
    !m_ReBar.AddBar(&m_ToolBarBasic) ||
    !m_ReBar.AddBar(&m_ToolBarMultiframe) ||
    !m_ReBar.AddBar(&m_wndDlgBar))
{
    TRACE0("Failed to create rebar\n");
    return -1;        // fail to create
}

// Create application status bar
if (!m_StatusBar.Create(this) ||
    !m_StatusBar.SetIndicators(indicators,
    sizeof(indicators)/sizeof(UINT)))
{
    TRACE0("Failed to create status bar\n");
    return -1;        // fail to create
}

// Insert image counter into multiframe toolbar
m_ToolBarMultiframe.MakeCStatic(ID_FRAME_NUMBER);
ShowFrameNumber(-1);

// Size progress bar
CClientDC dc(this);
SIZE size=dc.GetTextExtent(m_paneString);
int index=m_StatusBar.CommandToIndex(ID_PROGRESS_STATUS);
m_StatusBar.SetPaneInfo(index,ID_PROGRESS_STATUS, SBPS_NORMAL, size.cx);

// Drag-and-drop file support
DragAcceptFiles();

return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // Size the main frame window to the screen size and center it
    return CMDIFrameWnd::PreCreateWindow(cs);
}

////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CMDIFrameWnd::AssertValid();
}

```

```

void CMainFrame::Dump(CDumpContext& dc) const
{
    CMDIFrameWnd::Dump(dc);
}

#ifdef _DEBUG

////////////////////////////////////
// CMainFrame message handlers
void CMainFrame::OnUpdateProgressStatus(CCmdUI *pCmdUI)
{
    pCmdUI->Enable();
    pCmdUI->SetText(m_paneString);
}

/*****
 *
 *   Sets the number currently displayed in the frame edit toolbar
 *
 *****/
void CMainFrame::ShowFrameNumber(int n)
{
    CString st;
    if(n>0) st.Format("%d",n);
    else st.Format(" ");
    m_ToolBarMultiframe.SetInsertedControlText(st);
}

/*****
 *
 *   Exit from application - clean up
 *
 *****/
BOOL CMainFrame::DestroyWindow()
{
    // Close main window
    return CMDIFrameWnd::DestroyWindow();
}

/*****
 *
 *   Show/hide all toolbars
 *
 *****/
void CMainFrame::OnViewToolbar()
{
    m_showToolbars = !m_showToolbars;
    m_ReBar.GetReBarCtrl().ShowBand(0, m_showToolbars);
    m_ReBar.GetReBarCtrl().ShowBand(1, m_showToolbars);
    m_ReBar.GetReBarCtrl().ShowBand(2, m_showToolbars);
}

void CMainFrame::OnUpdateViewToolbar(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_showToolbars);
}

void CMainFrame::ShowMultiframeToolbar(bool show)
{
    m_ReBar.GetReBarCtrl().ShowBand(1, show);
}

/*****
 *
 *   Process dropdown toolbar buttons
 *
 *****/
void CMainFrame::OnToolbarDropDown(NMTOOLBAR *pnmtb, LRESULT *plr)
{
    m_ToolBarBasic.TrackDropDownMenu(pnmtb->iItem,this);
    m_ToolBarMultiframe.TrackDropDownMenu(pnmtb->iItem,this);
}

```



```

/*****
*
*   Drag-and-drop support
*
*****/
void CMainFrame::OnDropFiles(HDROP hDropInfo)
{
    // Find the number of files
    UINT nFiles = ::DragQueryFile(hDropInfo, (UINT)-1, NULL, 0);
    for(UINT iFile=0; iFile<nFiles; iFile++)
    {
        TCHAR szFileName[_MAX_PATH];
        ::DragQueryFile(hDropInfo,iFile,szFileName,_MAX_PATH);
        theApp.OpenDocumentFile(szFileName,true);
    }
    ::DragFinish(hDropInfo);
}

/*****
*
*   Switch logo animation on the menu bar
*
*****/
void CMainFrame::EnableLogoAnimation(bool enable)
{
    if(enable) m_Animate.Open(IDR_AVI_DCM);
    else      m_Animate.Stop();
}

/*****
*
*   Show dictionary availability on the status bar
*
*****/
void CMainFrame::SetDictionaryStatus(bool enabled)
{
    int index=m_StatusBar.CommandToIndex(ID_DICTIONARY_STATUS);
    CString stat = enabled ? "DICT ENABLED" : "DICT DISABLED";
    m_StatusBar.SetPaneText(index, stat);
}

```

```
// DICOMDocument.h: interface for the DICOMDocument class.
//
///////////////////////////////////////////////////////////////////

#ifndef AFX_DICOMDOCUMENT_H__INCLUDED_
#define AFX_DICOMDOCUMENT_H__INCLUDED_

#include "DICOMInfo.h" // Added by ClassView
#include "winmodules.h" // Added by ClassView
#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class DICOMDocument : public Array<ImageSeries> // Study
{
public:
    static const BYTE    FormatDICOMOriginal;
    static const BYTE    FormatDICOMModified;
    static const BYTE    FormatWINDOWSMultimedia;

    void                AddDICOMRecords(Array<DICOMRecord> &a);
    void                DisplayDICOMInfo();
    void                GetPixelSpacing(double &dx, double &dy);
    void                SetShowInfo(bool show);
    bool                GetShowInfo();
    bool                AddDDOFile(CString filename);
    bool                IsEmpty() { return m_DDO.IsEmpty(); }
    bool                SaveDICOM(CString filename="");
    bool                SaveDocument(CString fname, int format);
    bool                LoadFile(CString filename);
    bool                LoadDDO(DICOMDataObject &ddo, bool clone);
    int                GetNumberOfImages();
    int                GetCurrentImageIndex();
    CString             GetFilename() { return m_Filename; }
    CString             GetMRUAlias();
    Image*             GetImage(int n);
    inline ImageSeries* GetCurrentSeries()
    {
        return &m_ImageSeries;
        if(GetSize()<=0) return NULL; // empty
        if(m_CurrentSeries<0) m_CurrentSeries=0;
        if(m_CurrentSeries>=(int)GetSize()) m_CurrentSeries=GetUpperBound();
        return &(Get(m_CurrentSeries));
    };
    DICOMRecord*        GetDICOMRecordPtr() { return &m_FileRecord; };
    DICOMDocument();
    virtual ~DICOMDocument();

private:
    int                m_CurrentSeries;
    DICOMRecord        m_FileRecord;
    CString            m_Filename;
    PDU_Service        m_PDU;
    DICOMDataObject    m_DDO;
    DICOMInfo          m_InfoDialog;
    ImageSeries        m_ImageSeries;

    bool                InitializeDocument(DICOMDataObject& m_DDO,
                                           CString& m_Filename);
};

#endif // !defined(AFX_DICOMDOCUMENT_H__INCLUDED_)

```

//

////////////////////////////////////

```
#ifndef  DEBUG
```

```
#undef THIS_FILE
```

```
static char THIS_FILE[] = FILE ;
```

```
#define new DEBUG_NEW
```

```
#endif
```

////////////////////

```
// Construction/Destruction
```

////////////////////////////////////

```
const BYTE DICOMDocument::FormatDICOMOriginal = 0;
```

```
const BYTE DICOMDocument::FormatDICOMModified = 1;
```

```
const BYTE DICOMDocument::FormatWINDOWSMultimedia = 2;
```

[illegible]

```
{
    if(!theApp.app_RTC.IsEmpty())
```

```
m_PDU.AttachRTC(&(theApp.app_RTC), FALSE);
```

```
m CurrentSeries = -1;
```

```
DICOMDocument::~DICOMDocument()
```

$\left\{ \begin{array}{l} \text{1000} \\ \text{1000} \\ \text{1000} \\ \text{1000} \end{array} \right\}$

/\* \*\*\*\*

```
*1 2 3 Load DICOM Data from
```

\*≡ 1. A file

\* 2. Another DDO

```

*****

```

```
bool DICOMDocument::LoadFile(CString filename)
```

```
filename.TrimRight();    filename.TrimLeft();
if (filename == "")
{
```

```
AfxMessageBox("Cannot load: file name is empty",
              MB_OK|MB_ICONEXCLAMATION);
```

```
return false;
```

```
// Store filename
```

```
m Filename=filename;
```

```
// Load DDO, completely, using m PDU for RTC
```

```
m DDO.Reset();
```

```
if(!m_DDO.LoadFromFile((char*)(LPCSTR)(filename),false,&m_PDU))
```

```
AfxMessageBox("Cannot read DICOM from file\n"+filename,MB_OK|MB_ICONEXCLAMATION);
return false;
```

```
// Initialize data
```

```
if(!InitializeDocument(m_DDO, m_Filename)) return false;
```

// OK

```
return true;
```

```
bool DICOMDocument::LoadDDO(DICOMDataObject &ddo, bool clone)
```

$$\{ \quad \quad \quad VR * vr; \}$$

```
// Set m DDO
```

```
m DDO.Reset();
```

$\bar{i}\bar{f}$  (clone)

1

```

        if(!m_DDO.CloneFrom(&dco))
        {
            AfxMessageBox("Cannot load DICOM",MB_OK|MB_ICONEXCLAMATION);
            return false;
        }
    }
    else
    {
        while(vr=dco.Pop()) m_DDO.Push(vr);
    }
    // Set filename
    char ff[65];
    m_DDO.SuggestFileName(ff,65);
    m_Filename = theApp.app_DirectoryTmp+"/"+CString(ff);

    // Classify VRs
    theApp.app_RTC.RunTimeClass(&m_DDO);

    // Save m_DDO copy into m_Filename for consistency
    if( !m_DDO.SaveIntoFile((char*)(LPCSTR)m_Filename),true,&m_PDU) )
    {
        AfxMessageBox("Cannot save DICOM in file \n"+m_Filename,
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
    // Initialize data
    if(!InitializeDocument(m_DDO, m_Filename)) return false;
    return true;
}

/*****
*
* Initialize all document data from m_DDO and m_Filename
*
*****/
bool DICOMDocument::InitializeDocument(DICOMDataObject& m_DDO,
    CString& m_Filename)
{
    // Do we have anything inside the DICOM object ?
    if(m_DDO.IsEmpty())
    {
        AfxMessageBox("Empty or invalid DICOM object", MB_ICONEXCLAMATION|MB_OK);
        return false;
    }
    // Display, parse and hide DICOM Info
    //if(!m_InfoDialog.PopInfo(m_DDO, m_Filename)) return false;
    // Set file record
    m_FileRecord.SetRecord(m_DDO, (char*)(LPCSTR)m_Filename);
    // Parse image data into bitmaps
    m_ImageSeries.ReadDO(m_DDO, (char*)(LPCSTR)m_Filename);
    //Array<ImageSeries>::Add(m_ImageSeries);
    //m_CurrentSeries = Array<ImageSeries>::GetUpperBound();

    // If no images found, display info dialog
    if(GetNumberOfImages()<1) m_InfoDialog.DoModalless(m_DDO,m_Filename);
    return true;
}

/*****
*
* Save DICOM Data into a file
*
*****/
bool DICOMDocument::SaveDICOM(CString filename /*="" */)
{
    if(filename=="") filename=m_Filename;
    if(filename=="")
    {
        AfxMessageBox("Empty filename, cannot save",MB_OK|MB_ICONEXCLAMATION);
        return false;
    }

    // Save copy DO
    DICOMDataObject do_tmp;
    if(!do_tmp.CloneFrom(&m_DDO))

```

```

{
    AfxMessageBox("Cannot save DICOM data",MB_OK|MB_ICONEXCLAMATION);
    return false;
}
if(!GetCurrentSeries()->WriteDO(do_tmp))
{
    AfxMessageBox("Cannot save image data",MB_OK|MB_ICONEXCLAMATION);
    return false;
}
if(!do_tmp.SaveIntoFile((char*)(LPCSTR)(filename),false))
{
    AfxMessageBox("Cannot save DICOM in file \n"+filename,
        MB_OK|MB_ICONEXCLAMATION);
    return false;
}
return true;
}

/*****
*
*   Display DICOM Object
*
*****/
void DICOMDocument::DisplayDICOMInfo()
{
    m_InfoDialog.DoModalless(m_DDO,m_Filename);
}

/*****
*
*   Navigating in image series
*
*****/
Image* DICOMDocument::GetImage(int n)
{
    ImageSeries* pS = GetCurrentSeries();
    if(!pS) return NULL;
    else return pS->GetImage(n);
}

int DICOMDocument::GetNumberOfImages()
{
    ImageSeries* pS = GetCurrentSeries();
    if(!pS) return 0;
    else return pS->GetUpperBound()+1;
}

int DICOMDocument::GetCurrentImageIndex()
{
    ImageSeries* pS = GetCurrentSeries();
    if(!pS) return -1;
    else return pS->GetCurrentImageIndex();
}

/*****
*
*   Save the entire document in specified format
*
*****/
bool DICOMDocument::SaveDocument(CString fname, int format)
{
    bool success=true;
    CString err("Error saving the document");
    // Clean proposed file name
    fname.TrimLeft();    fname.TrimRight();
    if(fname=="")
    {
        AfxMessageBox("Cannot save: file name is empty",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    }

    // Save in requested format
    if(format==FormatDICOMOriginal)                // Save unchanged DICOM
    {

```

```

        if(fname!=m_Filename)
        {
            bool success=(CopyFile(m_Filename,fname,FALSE)!=0);
        }
    else if(format==FormatDICOMModified)        // Save modified DICOM
    {
        return SaveDICOM(fname);
    }
    else if(format==FormatWINDOWSMultimedia)    // Save DICOM as multimedia directory
    {
        // Create new directory
        if(!::CreateAndSetCurrentDirectory((char*)(LPCSTR)fname))
        {
            AfxMessageBox("Cannot access directory\n"+fname,
                MB_OK|MB_ICONEXCLAMATION);
            return false;
        }
        // Save all bitmaps
        success = GetCurrentSeries()->SaveAsImageFiles(fname);
        // Save DICOM demographics
        char info[_MAX_PATH];
        sprintf(info,"%s\\info.txt",fname);
        success = success && m_FileRecord.WriteIntoTextFile(info);

        // Save sound data
        /*
        if(m_SoundFileName.GetLength()>3)
        {
            try
            {
                CFile::Rename(m_SoundFileName, fname+"\\voice.wav");
            }
            catch (CFileException *e)
            {
                err.Format("Error saving sound file into %s",fname+fname+"voice.wav");
                AfxMessageBox(err);
                e->Delete();
                return FALSE;
            }
        }
        */
    else
    {
        err="Invalid format";    success=false;
    }
    if(!success)    AfxMessageBox(err,MB_OK|MB_ICONEXCLAMATION);
    Beep(500,100);
    return success;
}

/*****
 *
 *   Get physical spacing between image pixels
 *
 *****/
void DICOMDocument::GetPixelSpacing(double &dx, double &dy)
{
    GetCurrentSeries()->GetPixelSpacing(dx, dy);
}

/*****
 *
 *   Show image info on the images
 *
 *****/
void DICOMDocument::SetShowInfo(bool show)
{
    GetCurrentSeries()->SetShowSeriesImageInfo(show);
}
bool DICOMDocument::GetShowInfo()
{
    return GetCurrentSeries()->GetShowSeriesImageInfo();
}

```

```

}

/*****
*
*   Create an alias name for the MRU files list
*
*****/
CString DICOMDocument::GetMRUAlias()
{
    CString s;
    s.Format("%s, %s", m_FileRecord.GetPatientName(), m_FileRecord.GetPatientID());
    return s;
}

/*****
*
*   Include new series files
*
*****/
bool DICOMDocument::AddDDOFile(CString filename)
{
    ImageSeries* pS = GetCurrentSeries();
    if(pS) return pS->AddDDOFile(filename);
    else return false;
}

void DICOMDocument::AddDICOMRecords(Array<DICOMRecord> &a)
{
    if(a.GetSize()<=0) return;
    ImageSeries* pS = GetCurrentSeries();
    if(!pS)
    {
        LoadFile(a[0].GetFileName());
        a.RemoveAt(0);
        AddDICOMRecords(a); // recursion
        return;
    }
    else pS->AddDICOMRecords(a);
}

```

```

#if !defined(AFX_DICOMINFO_H_INCLUDED_)
#define AFX_DICOMINFO_H_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "../resource.h"
#include <dicom.hpp>

////////////////////////////////////
// DICOMInfo dialog

class DICOMInfo : public CDialog, public DICOMView
{
// Construction
public:
    void SetFilename(CString filename);
    void DoModeless();
    void DoModeless(DICOMObject& dob, CString filename);
    void Load(const char* str);
    void Load(DICOMObject &DO);
    bool PopInfo(DICOMObject& dob, CString& filename);
    bool LoadFile(char* filename);
    DICOMInfo(RTC* rtc, CString temp_dir, CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{AFX_DATA(DICOMInfo)
    enum { IDD = IDD_DIALOG_DICOMINFO };
    //}{AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(DICOMInfo)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}{AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(DICOMInfo)
    afx_msg void OnOK();
    afx_msg void OnCancel();
    afx_msg void OnDropFiles(HDROP hDropInfo);
    virtual BOOL OnInitDialog();
    //}{AFX_MSG
    afx_msg void OnClose();
    DECLARE_MESSAGE_MAP()

private:
    CString m_Filename, m_BackupFile, m_TemporaryDirectory;
    const CString m_DropFile;
    CListCtrl m_List;

    void SaveToFile();
    void ResetBackup();
    bool OpenFromFile();
};

//{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DICOMINFO_H_INCLUDED_)

```



```
// dicominfo.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "dicominfo.h"
#include <io.h>
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// DICOMInfo dialog
```

```
DICOMInfo::DICOMInfo(RTC* rtc, CString temp_dir, CWnd* pParent /*=NULL*/)
: CDialog(DICOMInfo::IDD, pParent), m_DropFile("")
```

```
{
    //{AFX_DATA_INIT(DICOMInfo)
    m_Filename = _T("");
    //}AFX_DATA_INIT
    // Set RTC
    if(!rtc->IsEmpty()) AttachRTC(rtc);
    // Set temporary directory
    m_TemporaryDirectory=temp_dir;
    m_TemporaryDirectory.Replace('/', '\\');
    m_TemporaryDirectory.TrimRight(" \\");
    // Set backup file name
    m_BackupFile="";
}
```

```
void DICOMInfo::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{AFX_DATA_MAP(DICOMInfo)
    DDX_Control(pDX, IDC_LIST, m_List);
    DDX_Text(pDX, IDC_FILENAME, m_Filename);
    //}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(DICOMInfo, CDialog)
```

```
//{AFX_MSG_MAP(DICOMInfo)
ON_WM_DROPFILES()
//}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// DICOMInfo message handlers
```

```
/*
*****
*
* Overriden virtuals from DICOMView
*
*****
*/
```

```
void DICOMInfo::Load(const char *str)
```

```
{
    int n,m;
    CString cs=CString(str);
    cs.TrimRight(); cs.TrimLeft();
    if(cs=="") return;
    // Parse VR string
    CString tag; n=cs.Find(",");
    if(n<0) tag="";
    else { tag=cs.Left(n+1); cs=cs.Mid(n+2); }
    tag.TrimRight();
```

```
CString length; n=cs.Find("bytes,");
if(n<0) length="";
else { length=cs.Left(n+5); cs=cs.Mid(n+6); }
length.TrimLeft(); length.TrimRight();
```

```

CString vr;
n=cs.Find("VR=");
if(n<0) vr="";
else {   vr=cs.Mid(n+4,2);       cs=cs.Mid(n+7); }
vr.TrimLeft();  vr.TrimRight();

CString value, description;
n=cs.Find("<"); m=cs.Find("[",max(n,0));
if(n<0) value="";
else
{
    if(m>n)
    {
        value=cs.Mid(n,m-n);    description=cs.Mid(m+2);
        description.TrimRight(" ");
    }
    else
    {
        value=cs.Mid(n);        description="";
    }
}
value.TrimLeft();    value.TrimRight();
description.TrimLeft(); description.TrimRight();

UINT nItem=m_List.InsertItem(m_List.GetItemCount(),tag);
m_List.SetItem(nItem,1,LVIF_TEXT,length,0,0,0,0);
m_List.SetItem(nItem,2,LVIF_TEXT,vr,0,0,0,0);
m_List.SetItem(nItem,3,LVIF_TEXT,value,0,0,0,0);
m_List.SetItem(nItem,4,LVIF_TEXT,description,0,0,0,0);
}

void DicomInfo::Load(DICOMObject &DO)
{
    bool top_level=(m_SequenceLevel==0);
    if(top_level)
    {
        GetDlgItem(IDC_EDITNUM)->SetWindowText("No elements found");
        m_List.DeleteAllItems();
        BeginWaitCursor();
        if(OpenFromFile()) return;
    }
    DicomView::Load(DO);
    if(top_level)
    {
        CString num;    num.Format("%d",m_numElements);
        GetDlgItem(IDC_EDITNUM)->SetWindowText(num);
        for(int i=0; i<=4; i++)
        {
            m_List.SetColumnWidth(i,LVSCW_AUTOSIZE);
        }
        EndWaitCursor();
        SaveToFile();
    }
}

bool DicomInfo::LoadFile(char *filename)
{
    BeginWaitCursor();
    m_List.DeleteAllItems();
    bool success = true;
    CString fn(filename);    fn.TrimRight();
    if(fn==m_Filename && OpenFromFile()) success=true;
    else success=DicomView::LoadFile(filename);
    if(!success) GetDlgItem(IDC_EDITNUM)->SetWindowText(
        "Cannot load this file");

    SetFilename(fn);
    EndWaitCursor();
    return success;
}

```

```

/*****
*
*   Modeless display and distruction
*
*****/

```

```

void DicomInfo::DoModeless()
{
    if(this->GetSafeHwnd()) return;
    Create(IDD_DIALOG_DICOMINFO, NULL);
    // Always display on top
    SetWindowPos(&wndTopMost, 0,0,0,0,SWP_NOSIZE | SWP_SHOWWINDOW );
    SetWindowText("DICOM Header");
    ShowWindow(SW_SHOWNORMAL);
}

void DicomInfo::DoModeless(DicomObject &dob, CString filename)
{
    if(this->GetSafeHwnd()) return;
    DoModeless();
    SetFilename(filename);
    Load(dob);
}

void DicomInfo::OnOK() { DestroyWindow(); }
void DicomInfo::OnCancel() { DestroyWindow(); }
void DicomInfo::OnClose() { DestroyWindow(); }
bool DicomInfo::PopInfo(DicomObject &dob, CString &filename)
{
    DoModeless(dob, filename);
    bool non_empty = (m_numElements>0);
    OnClose();
    if(!non_empty)
    {
        AfxMessageBox("Invalid DICOM format", MB_OK|MB_ICONEXCLAMATION);
    }
    return non_empty;
}

/*****
 *
 * Dragging files
 *
 *****/
void DicomInfo::OnDropFiles(HDROP hDropInfo)
{
    // Find the number of files
    UINT nFiles = ::DragQueryFile(hDropInfo, (UINT)-1, NULL, 0);
    if(nFiles>1)
    {
        AfxMessageBox("Cannot display multiple files."
            "\nSelect one file and try again.",
            MB_OK|MB_ICONEXCLAMATION);
        return;
    }
    TCHAR szFileName[256];
    ::DragQueryFile(hDropInfo, 0, szFileName, 256);
    CString temp=m_BackupFile; m_BackupFile=m_DropFile;
    LoadFile(szFileName);
    m_BackupFile=temp;
    ::DragFinish(hDropInfo);
}

/*****
 *
 * Set filename
 *
 *****/
void DicomInfo::SetFilename(CString filename)
{
    m_Filename=filename; m_Filename.TrimRight();
    UpdateData(FALSE);
}

BOOL DicomInfo::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Element list columns
    m_List.InsertColumn(0, "Element Tag", LVCFMT_CENTER, 60, 0);

```

```

    m_List.InsertColumn(1,"Length", LVCFMT_CENTER,80,0);
    m_List.InsertColumn(2,"VR", LVCFMT_CENTER,80,0);
    m_List.InsertColumn(3,"Value", LVCFMT_LEFT,100,0);
    m_List.InsertColumn(4,"Description", LVCFMT_LEFT,80,0);
    // Element list: Force entire row selection
    m_List.SendMessage( LVM_SETEXTENDEDLISTVIEWSTYLE, 0,LVS_EX_FULLROWSELECT );
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

```

```

/*****
*
* Persistent Storage
*
*****/

```

```

void DICOMInfo::SaveToFile()
{

```

```

    if(m_BackupFile==m_DropFile) return;
    if(m_BackupFile=="") // was not initialized
    {
        char curdir[MAX_PATH+1];
        GetCurrentDirectory(MAX_PATH,curdir);
        SetCurrentDirectory(m_TemporaryDirectory);
        char name_template[13] = "_info_XXXXXX";
        bool success = (mktemp(name_template) != NULL);
        SetCurrentDirectory(curdir);
        if(!success) return;
        m_BackupFile=m_TemporaryDirectory+"\\\\"+CString(name_template);
    }

```

```

    int i, n;
    CString tag, length, vr, value, description;
    try
    {
        CFile db_file(m_BackupFile, CFile::modeCreate | CFile::modeWrite);
        CArchive ar(&db_file, CArchive::store);
        n=m_List.GetItemCount();
        ar<<n;
        for(i=0; i<n; i++)
        {
            tag=m_List.GetItemText(i,0);
            length=m_List.GetItemText(i,1);
            vr=m_List.GetItemText(i,2);
            value=m_List.GetItemText(i,3);
            description=m_List.GetItemText(i,4);
            ar<<tag<<length<<vr<<value<<description;
        }
        ar.Close();
    }
    catch(CException* e)
    {
        e->Delete();
        ResetBackup();
    }
}

```

```

bool DICOMInfo::OpenFromFile()
{

```

```

    if(m_BackupFile==" " || m_BackupFile==m_DropFile) return false;
    int i, n;
    UINT nItem;
    CString tag, length, vr, value, description;
    try
    {
        CFile db_file(m_BackupFile, CFile::modeRead);
        CArchive ar(&db_file, CArchive::load);
        ar>>n;
        m_List.DeleteAllItems();
        for(i=0; i<n; i++)
        {
            ar>>tag>>length>>vr>>value>>description;
            nItem=m_List.InsertItem(m_List.GetItemCount(),tag);
            m_List.SetItem(nItem,1,LVIF_TEXT,length,0,0,0,0);
            m_List.SetItem(nItem,2,LVIF_TEXT,vr,0,0,0,0);
            m_List.SetItem(nItem,3,LVIF_TEXT,value,0,0,0,0);

```

```

        m_List.SetItem(nItem,4,LVIF_TEXT,description,0,0,0,0);
    }
    ar.Close();
    m_numElements=n;
    CString num;    num.Format("%d",m_numElements);
    GetDlgItem(IDC_EDITNUM)->SetWindowText(num);
    for(i=0; i<=4; i++)
    {
        m_List.SetColumnWidth(i,LVSCW_AUTOSIZE);
    }
    return true;
}
catch(CException* e)
{
    e->Delete();
    ResetBackup();
    return false;
}
}

```

```

/*****

```

```

*   Reset the backup file

```

```

*****/

```

```

void DICOMInfo::ResetBackup()

```

```

{
    DeleteFile(m_BackupFile);
    m_BackupFile="";
}

```

```

// Image.h: interface for the Image class.
//
/////////////////////////////////////////////////////////////////

#if !defined(AFX_IMAGE_H_INCLUDED_)
#define AFX_IMAGE_H_INCLUDED_

#include "ScreenMap.h"
#include "Palette.h"
#include "../StdAfx.h" // Added by ClassView

#define COMPARE_CORRELATION 0
#define COMPARE_ABSOLUTE 1

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
class Image : public CObject
{
public:
    bool m_UpdateBitmap;
    bool m_RGB;
    BYTE m_EdgeThreshold;
    static const BYTE
        DisplayNormal, DisplayHidden, DisplaySelected;
    long m_maxPixelValue;
    Palette* m_pPal;
    ScreenMap m_ScreenMap;

    void SetImageData(DICOMRecord* dr, Point2D* pspace, int* index);
    void SetImageRectZoom(double zoom);
    void SetImageRectOffset(double dx, double dy);
    void SetDisplayStatus(BYTE stat);
    void LinkToPalette(Palette* pPal);
    void GetSubimage(Image* sub, const int xleft, const int ytop);
    void Palette_to_Pixels();
    void ResetPixels(bool current_to_safe);
    void SetBrightness (int b);
    void TR_Flip (bool horizontal);
    void SetShowImageInfo(bool show) { m_ShowImageInfo = show; };
    void GetStatistics(long &avg, long &sigma2, long &count,
        int x0, int x1, int y0, int y1);
    void GetStatistics(long &avg, long &sigma2, long &count)
    {
        GetStatistics(avg, sigma2, count, 0, m_Width, 0, m_Height);
    };
    inline void SetPixel(unsigned int x, unsigned int y, long p);
    inline void SetPixelFromLum(unsigned int x, unsigned int y, long p);
    inline void SetPixel(unsigned long n, long p);
    bool ContainsScreenPoint(CPoint& p);
    bool GetShowImageInfo() { return m_ShowImageInfo; };
    bool CloneFrom(Image* img);
    bool WriteToFile(CString fname);
    bool SetPalette(long* color_map, long color_map_size, bool show_progress=true);
    bool SetPalette(int px, int py, int pxmax, int pymax);
    bool CreateImage(int xwidth, int xheight,
        int bytes_per_pixel, Palette* pPal=NULL);
    bool TR_SetUniformLuminance(CRect r, bool show_progress=true);
    bool TR_SetUniformLuminance(bool show_progress=true);
    bool TR_PixelNeighborhood(char mask_type, bool show_progress);
    bool TR_PixelExp();
    bool TR_PixelLog();
    bool TR_GammaCorrection(double gamma);
    bool TR_HistStretch(BYTE percent, bool show_progress=true);
    bool TR_HistStretch(int amin, int amax, int bmin, int bmax,
        bool show_progress=true);
    bool TR_HistEqualize(bool show_progress=true);
    bool TR_Negate();
    bool TR_Rotate(int degrees);
    bool DisplayDIB(CDC *pDC, CRect crectScreenArea, CRect crectImageArea,
        CPoint pScroll=CPoint(0,0), bool optimize=true);
    bool DisplayDIB(CDC *pDC, CPoint pScroll);
    bool DisplayDIB(CDC *pDC);

```

```

BYTE*      GetPixelBytes(UINT32& data_size);
inline BYTE GetDisplayStatus() { return m_DisplayStatus; };
int        CompareToDICOMRecord(DICOMRecord & dr, BYTE lev);
inline int  GetHeight() { return m_Height; };
inline int  GetWidth() { return m_Width; };
inline int  GetBytesPerPixel() { return m_Bytes_per_Pixel; }
inline int  GetBytesInRow() { return m_Height*m_Bytes_per_Pixel; }
long        TR_Fractal(const int x, const int y);
inline long GetPixel(unsigned int x, unsigned int y);
inline long GetPixel(unsigned long n);
inline long GetLuminance(int x, int y);
inline long GetSmoothedLuminance(int x, int y);
double      Compare(int comp_type, CPoint topleft, long **pattern,
                    int pat_w, int pat_h, long pat_avr);

inline double
    GetZoom() { return m_ScreenMap.GetZoom(); };
CPoint      TR_FindSimilarRegion(const CRect r0);
Image(bool undo=true);
~Image();

```

```
private:
```

```

bool        m_ReleasePalette;
bool        m_DisplayFailed;
bool        m_ShowImageInfo;
BYTE        m_DisplayStatus;
BYTE*       m_Pixels;
unsigned int m_numColors;
int         m_Index;
int         m_Height, m_Width;
int         m_Bytes_per_Pixel;
long        m_numPixelBytes;
unsigned long m_numPixels;
static unsigned long
    m_UndoFileCount;
Point2D     m_PixelSpacing;
CString     m_UndoFile;
HBITMAP     m_Bitmap;
CStatusBar* m_MainStatusBar;
BITMAPFILEHEADER m_bmpFHeader;
LPBITMAPINFO m_bmpInfo;
DICOMRecord m_DICOMRecord;

void        DisplayImageInfo(CDC* pDC);
void        FormatImageInfo(CString& info);
void        Get_Pixel_minmax(long& pmin, long& pmax);
void        DeleteImageData();
bool        SerializeImage(FILE* fp, bool is_loading);
inline unsigned long
    MapPixel(unsigned int x, unsigned int y);
long        TR_DeNoise(const int x, const int y);
long        TR_Sharp(const int x, const int y);
long        TR_Smooth(const int x, const int y);
long        TR_Sobel(const int x, const int y);
HBITMAP     DIB_to_DDB(CDC *pDC);
};

```

```

/*****
*
*   Get pixel functions
*
*****/
inline long Image::GetPixel(unsigned int x, unsigned int y)
{
    return GetPixel(MapPixel(x,y));
}

```

```

}
inline long Image::GetPixel(unsigned long n)
{
    if(n>=m_numPixels) n=m_numPixels-1;
    n *= m_Bytes_per_Pixel; // array index
    // Color ?
    if(m_RGB) return RGB(m_Pixels[n+2],m_Pixels[n+1],m_Pixels[n]);
    // Grayscale
    switch(m_Bytes_per_Pixel)
    {
        case 1: return m_Pixels[n];
        case 2: return ((UINT16*)m_Pixels)[n];
        case 3: return (m_Pixels[n+1] + (((long)m_Pixels[n+1])<<8)
                        +(((long)m_Pixels[n+2])<<16) );
        case 4: return ((UINT32*)m_Pixels)[n];
    }
    return m_Pixels[n];
}

/*****
 *
 *   Set pixel functions
 *
 *****/
inline void Image::SetPixel(unsigned int x, unsigned int y, long p)
{
    m_UpdateBitmap=true;
    SetPixel(MapPixel(x,y),p);
}
inline void Image::SetPixelFromLum(unsigned int x, unsigned int y, long p)
{
    m_UpdateBitmap=true;
    long n=MapPixel(x,y);
    if(m_RGB) // color image
    {
        m_Pixels[n] = (BYTE)p;
        m_Pixels[n+1]=(BYTE)p;
        m_Pixels[n+2]=(BYTE)p;
        return;
    }
    else SetPixel(n,p);
}
inline void Image::SetPixel(unsigned long n, long p)
{
    m_UpdateBitmap=true;
    if(n>=m_numPixels) n=m_numPixels-1;
    // Color ?
    if(m_RGB)
    {
        m_Pixels[n]=GetBValue(p);
        m_Pixels[n+1]=GetGValue(p);
        m_Pixels[n+2]=GetRValue(p);
        return;
    }
    // Grayscale
    if(p>m_maxPixelValue) p=m_maxPixelValue;
    else if(p<0) p=0;
    switch(m_Bytes_per_Pixel)
    {
        case 1: m_Pixels[n]=(BYTE)p;
                break;
        case 2: ((UINT16*)m_Pixels)[n] = (UINT16)p;
                break;
        case 3:
                m_Pixels[n+2]=(BYTE) (p>>16);
                m_Pixels[n+1]=(BYTE) ((p>>8)&255);
                m_Pixels[n]=(BYTE) (p&255);
                break;
        case 4: ((UINT32*)m_Pixels)[n] = (UINT32)p;
                break;
    }
    return;
}

```



```

/*****
*
*   Returns luminance (brightness) of pixel (x,y)
*
*****/
inline long Image::GetLuminance(int x, int y)
{
    if(x<=0)    x=0;        if(y<0) y=0;
    if(m_RGB)
    {
        long i=MapPixel(x,y);
        return (long)((m_Pixels[i]+m_Pixels[i+1]+m_Pixels[i+2])/3);
    }
    else return GetPixel(x,y);
}

inline long Image::GetSmoothedLuminance(int x, int y)
{
    if(x<0) x=0;
    if(y<0) y=0;
    long p=TR_Smooth(x,y);
    if(m_RGB)
    {
        return (long)((GetRValue(p)+GetGValue(p)+GetBValue(p))/3);
    }
    else return p;
}

#endif // !defined(AFX_IMAGE_H_INCLUDED_)

```

```
// Image.cpp: implementation of the Image class.
```

```
//  
////////////////////////////////////
```

```
#include "stdafx.h"  
#include "..\DCM.h"  
#include "Image.h"  
#include "..\FindRegion.h"
```

```
#ifdef _DEBUG  
#undef THIS_FILE  
static char THIS_FILE[]=__FILE__;  
#define new DEBUG_NEW  
#endif
```

```
////////////////////////////////////  
// Construction/Destruction  
////////////////////////////////////
```

```
const BYTE Image::DisplayNormal = 0;  
const BYTE Image::DisplayHidden = 1;  
const BYTE Image::DisplaySelected = 2;  
unsigned long Image::m_UndoFileCount=0;  
Image::Image(bool undo /*=true*/)
```

```
{  
    m_Pixels=NULL;  
    m_bmpInfo=NULL;  
    m_Bitmap=NULL;  
    m_pPal=NULL;  
    m_ReleasePalette = true;  
    if(undo) m_UndoFile = "+";  
    else m_UndoFile = "-";  
    m_ShowImageInfo = false;  
    m_DisplayStatus = DisplayNormal;  
    m_Index=0;  
}
```

```
Image::~Image()
```

```
{  
    DeleteImageData();  
    if((m_UndoFile != "-") && (m_UndoFile != "+")) DeleteFile(m_UndoFile);  
}
```

```
void Image::DeleteImageData()
```

```
{  
    if(m_Pixels) { delete [] m_Pixels; m_Pixels=NULL; }  
    if(m_bmpInfo) { delete [] m_bmpInfo; m_bmpInfo=NULL; }  
    if(m_Bitmap) DeleteObject(m_Bitmap);  
    if(m_ReleasePalette)  
    {  
        if(m_pPal) delete m_pPal;  
        m_pPal=0;  
    }  
}
```

```
void Image::SetImageData(DICOMRecord *dr, Point2D *pspace, int *index)
```

```
{  
    if(dr) m_DICOMRecord = (*dr);  
    if(pspace) m_PixelSpacing = (*pspace);  
    if(index) m_Index = (*index);  
}
```

```
/*  
*****  
* Initialize image pixel arrays. Always call after image construction  
*  
*****  
bool Image::CreateImage(int xwidth, int xheight, int bytes_per_pixel,  
    Palette* pPal /*=NULL*/)
```

```
{  
    // Clear if anything existed  
    DeleteImageData();  
    // Find number of bytes per pixel with the current display  
    if(bytes_per_pixel>=1 && bytes_per_pixel<=3)  
    {
```

```

    m_Bytes_per_Pixel= bytes_per_pixel;
}
else return false; // out of memory
int device_bpp = (theApp.app_Metheus ? 2 : 1); // bytes per pixel available
if (device_bpp<m_Bytes_per_Pixel) m_Bytes_per_Pixel=device_bpp;
m_RGB=false; // no color images so far

// Set image parameters
if(xwidth>8)    m_Width=8*(xwidth/8);        else m_Width=8;
if(xheight>8)   m_Height=8*(xheight/8);       else m_Height=8;

if(m_Bytes_per_Pixel==1)    m_numColors=256;
else                        m_numColors=0;

m_numPixels=((long)m_Width)*m_Height; // total number of pixels
m_numPixelBytes=m_numPixels*m_Bytes_per_Pixel;
if(!m_RGB) m_maxPixelValue= min(4095, (1<<(8*m_Bytes_per_Pixel))-1);
else      m_maxPixelValue=255;
m_EdgeThreshold=max(1,m_maxPixelValue/20); // 5% of the total scale

// Set bitmap infoheader
m_bmpInfo=(LPBITMAPINFO)new BYTE[sizeof(BITMAPINFOHEADER)
+m_numColors*sizeof(RGBQUAD)];
if(!m_bmpInfo) return false; // out of memory for bitmapinfo
m_bmpInfo->bmiHeader.biSize=sizeof(BITMAPINFOHEADER); //fixed
m_bmpInfo->bmiHeader.biWidth=m_Width;
m_bmpInfo->bmiHeader.biHeight=m_Height;
m_bmpInfo->bmiHeader.biPlanes=1; //fixed
m_bmpInfo->bmiHeader.biBitCount=8*m_Bytes_per_Pixel;
m_bmpInfo->bmiHeader.biCompression=BI_RGB; //fixed
m_bmpInfo->bmiHeader.biSizeImage=0; //fixed
m_bmpInfo->bmiHeader.biXPelsPerMeter=0; //fixed
m_bmpInfo->bmiHeader.biYPelsPerMeter=0; //fixed
m_bmpInfo->bmiHeader.biClrUsed=0;
m_bmpInfo->bmiHeader.biClrImportant=0; //fixed, normally 0
// Set bitmap palette
for(unsigned int j=0; j<m_numColors; j++)
{
    m_bmpInfo->bmiColors[j].rgbRed=j;
    m_bmpInfo->bmiColors[j].rgbGreen=j;
    m_bmpInfo->bmiColors[j].rgbBlue=j;
    m_bmpInfo->bmiColors[j].rgbReserved=0;
}

// Set default screen map
m_ScreenMap.Initialize(CRect(0,0,m_Width,m_Height));
m_ScreenMap.crScreen.OffsetRect(50,0);

// Set default bitmap fileheader (used only for file I/O)
m_bmpFHeader.bfType=('M'<<8)|'B'; //fixed
m_bmpFHeader.bfSize=sizeof(BITMAPFILEHEADER)+sizeof(BITMAPINFO)
+m_Width*m_Height*m_numColors;
m_bmpFHeader.bfReserved1=0; //fixed
m_bmpFHeader.bfReserved2=0; //fixed
m_bmpFHeader.bfOffBits=sizeof(BITMAPINFOHEADER)+sizeof(BITMAPFILEHEADER)
+m_numColors*sizeof(RGBQUAD);

// Allocate pixel array, set display pixels to 0
try
{
    m_Pixels=new BYTE[m_numPixelBytes];
}
catch(...)
{
    AfxMessageBox("Image error: out of memory", MB_OK|MB_ICONEXCLAMATION);
    return false;
}
if(!m_Pixels)
{
    AfxMessageBox("Image error: out of memory", MB_OK|MB_ICONEXCLAMATION);
    return false;
}
else    memset(m_Pixels,0,m_numPixelBytes);

```

```

// Grab main window status bar
CMDIFrameWnd* main_frame=(CMDIFrameWnd*)AfxGetMainWnd();
m_MainStatusBar=(CStatusBar*)(main_frame->GetMessageBar());

// Set logical palette
if(!pPal) // create independent palette
{
    m_pPal=new Palette(theApp.app_Metheus,m_maxPixelValue,m_RGB);
    m_ReleasePalette=true;
}
else // link to existing palette
{
    m_pPal=pPal;
    m_ReleasePalette=false;
}

m_DisplayFailed=false; m_UpdateBitmap=true; m_Bitmap=NULL;

return true;
}

/*****
*
* Set image palette (will be set on devices supporting palettes)
* Must be called for 8-bit bitmaps
*
*****/
//Set palette to specified color map array
bool Image::SetPalette(long* color_map, long color_map_size, bool show_progress)
{
    long i, r, g, b, p;

    if(m_pPal->p_active) // if using logical palettes
    {
        return m_pPal->SetPalette(color_map, color_map_size);
    }
    else // directly reset pixel values
    {
        // Display progress control in the main frame status bar
        if(show_progress) theApp.ShowProgress(1,"Modifying pixel values ...");
        // Backup old pixel values
        ResetPixels(true);
        // Remap pixel values
        if(m_RGB)
        {
            for(i=0; i<(long)m_numPixels; i++)
            {
                if(show_progress && i%100==0) theApp.ShowProgress((99*i)/m_numPixels);
                p=GetPixel(i);
                r=GetRValue(p); if(r>=color_map_size) r=color_map_size-1;
                g=GetGValue(p); if(g>=color_map_size) g=color_map_size-1;
                b=GetBValue(p); if(b>=color_map_size) b=color_map_size-1;
                SetPixel(i,RGB(color_map[r],color_map[g],color_map[b]));
            }
        }
        else // greyscale
        {
            for(i=0; i<(long)m_numPixels; i++)
            {
                if(show_progress && i%100==0) theApp.ShowProgress((99*i)/m_numPixels);
                p=GetPixel(i); if(p>=color_map_size) p=color_map_size-1;
                SetPixel(i,color_map[p]);
            }
        }
        return true;
    }
}

bool Image::SetPalette(int px, int py, int pxmax, int pymax) // set palette with respect to the
mouse position
{

```

```

int offset=((long)m_maxPixelValue/3)*(2*px-pxmax)/pxmax;
double x=(2.0*py)/pymax;
x=1+2*(x-1)*(x-1)*(x-1);
return m_pPal->SetPalette(offset,x);
}

```

```

/*****
 *
 *   Plot DIB pixels
 *
 *****/
bool Image::DisplayDIB(CDC *pDC, CRect crectScreenArea, CRect crectImageArea,
                      CPoint pScroll, bool optimize)
{
    if(m_DisplayStatus == DisplayHidden)    return true;    // no display for hidden
    UINT palUsage;

    if(m_DisplayFailed) return false;
    if(m_pPal->p_active && m_pPal->LoadPalette(pDC,m_RGB))
    {
        palUsage=DIB_PAL_COLORS;
    }
    else palUsage=DIB_RGB_COLORS;

    // Printing ?
    if(pDC->IsPrinting())
    {
        optimize=false;
        int list_width  = pDC->GetDeviceCaps(HORZRES);
        int list_height = pDC->GetDeviceCaps(VERTRES);
        double mag = 0.9*min( (double)(list_width/GetWidth()),
                             (double)(list_height/GetHeight()));
        int print_width =(int)(mag*GetWidth());
        int print_height=(int)(mag*GetHeight());
        crectScreenArea=CRect(
            CPoint((list_width-print_width)/2,(list_height-print_height)/2),
            CSize(print_width,print_height));
        crectImageArea =CRect(CPoint(0,0),CSize(GetWidth(),GetHeight()));
        pScroll = CPoint(0,0);
    }

    /* Optimize drawing regions */
    CRect rcClip, rcDraw;
    pDC->GetClipBox(rcClip);
    rcClip.NormalizeRect();
    if(optimize)
    {
        CRect rcDIB;

        if(rcClip.IsRectEmpty())    return true;
        rcClip.InflateRect(4,4);
        rcDraw.IntersectRect(rcClip,crectScreenArea);
        rcDraw.NormalizeRect();
        if(rcDraw.IsRectEmpty())    return true;

        rcDIB=m_ScreenMap.Screen_to_Image(rcDraw);
        if(rcDIB.IsRectEmpty())    return true;

        crectScreenArea.CopyRect(rcDraw);
        if(theApp.app_Metheus)    crectScreenArea.OffsetRect(-pScroll);
        crectImageArea.CopyRect(rcDIB);
    }

    /* Correct image area */
    if(crectImageArea.left<0)    crectImageArea.OffsetRect(-crectImageArea.left,0);
    else if (crectImageArea.right>m_Width)
        crectImageArea.OffsetRect(m_Width-crectImageArea.right,0);
    if(crectImageArea.top <0)    crectImageArea.OffsetRect(0,-crectImageArea.top);
    else if (crectImageArea.bottom>m_Height)
        crectImageArea.OffsetRect(0,m_Height-crectImageArea.bottom);
}

```

```

/* Set screen (destination) and bitmap (source) areas */
long xDest=crectScreenArea.TopLeft().x; long yDest=crectScreenArea.TopLeft().y;
long wDest=crectScreenArea.Width();      long hDest=crectScreenArea.Height();
long xBmp=crectImageArea.TopLeft().x;
long yBmp;
if(theApp.app_Metheus) yBmp=crectImageArea.TopLeft().y;
else
{
    yBmp=m_Height-crectImageArea.TopLeft().y-crectImageArea.Height();
}
long wBmp=crectImageArea.Width();      long hBmp=crectImageArea.Height();

/* Display as DIB */
if(wDest==0 || hDest==0 || wBmp==0 || hBmp==0) return true;
if(!theApp.app_Metheus)
{
    m_DisplayFailed= (StretchDIBits( pDC->GetSafeHdc(),
                                     xDest, yDest, wDest, hDest, xBmp, yBmp, wBmp, hBmp,
                                     m_Pixels, m_bmpInfo, palUsage, SRCCOPY)<=0);
}
else // Metheus
{
    m_DisplayFailed=true;
    if(m_UpdateBitmap)
    {
        if(m_Bitmap)
            MetheusDeleteCompatibleBitmap(pDC->GetSafeHdc(),m_Bitmap);
        m_Bitmap = MetheusCreateCompatibleBitmap(pDC->GetSafeHdc(),GetWidth(), GetHeight());
        if(MetheusLoadImageFromData(pDC->GetSafeHdc(), m_Bitmap,
            theApp.app_DynamicPaletteStart,m_Pixels, GetWidth(), GetHeight(),
            GetWidth()*m_Bytes_per_Pixel,8*m_Bytes_per_Pixel,
            0, 0, GetWidth(), GetHeight(), 0,0)==FALSE)
        {
            AfxMessageBox("Cannot create image bitmap", MB_OK | MB_ICONEXCLAMATION);
            return false;
        }
        m_DisplayFailed=
            (MetheusStretchBltImage(pDC->GetSafeHdc(), NULL,
                                   xDest, yDest, wDest, hDest,
                                   m_Bitmap,
                                   xBmp, yBmp, wBmp, hBmp, SRCCOPY)==FALSE);
    }
}
/* Display selection frame */
if(m_DisplayStatus == DisplaySelected)
{
    CRect r2 = m_ScreenMap.crScreen;
    pDC->DrawEdge(r2,EDGE_BUMP, BF_RECT);
}

/* Display image info, if needed */
if(m_ShowImageInfo) DisplayImageInfo(pDC);

/* Any display errors ?? */
if(m_DisplayFailed)
{
    DWORD ecode=GetLastError();
    CString estr;
    estr.Format(" GDI error code: %ld\n Please reload the image", ecode);
    AfxMessageBox(estr, MB_OK | MB_ICONEXCLAMATION);
}
m_UpdateBitmap=false;
return (!m_DisplayFailed);
}

bool Image::DisplayDIB(CDC *pDC)
{
    return DisplayDIB(pDC,m_ScreenMap.crScreen,m_ScreenMap.crImage, CPoint(0,0), true);
}

bool Image::DisplayDIB(CDC *pDC, CPoint pScroll)
{
    return DisplayDIB(pDC,m_ScreenMap.crScreen,m_ScreenMap.crImage, pScroll, true);
}

```

```

/*****
*
*   Display image caption over the image
*
*****/
void Image::DisplayImageInfo(CDC *pDC)
{
    CString info;
    FormatImageInfo(info);
    if(pDC->IsPrinting())
    {
        pDC->SetMapMode(MM_LOENGLISH);
        CFont *oldcf = pDC->SelectObject(&theApp.app_MediumFont);
        pDC->SetTextColor(RGB(0,0,0));
        pDC->TextOut(0,0,info);
        pDC->SelectObject(oldcf);
        pDC->SetMapMode(MM_TEXT);
    }
    else
    {
        pDC->SetMapMode(MM_TEXT);
        CFont *oldcf = pDC->SelectObject(&theApp.app_SmallFont);
        // Find caption rectangle sizes
        CRect r=CRect(0,0,10,10);
        r.OffsetRect(m_ScreenMap.crScreen.TopLeft()+CSize(5,5));
        pDC->DrawText(info,r,DT_CALCRECT);
        r |= m_ScreenMap.crScreen;
        // Find out the best caption color
        COLORREF c1 = pDC->GetPixel(r.TopLeft());
        COLORREF c2 = pDC->GetPixel(r.BottomRight());
        COLORREF c3 = pDC->GetPixel(r.CenterPoint());
        COLORREF c = RGB(
            (GetRValue(c1)+GetRValue(c2)+GetRValue(c3))/3,
            (GetGValue(c1)+GetGValue(c2)+GetGValue(c3))/3,
            (GetBValue(c1)+GetBValue(c2)+GetBValue(c3))/3);
        BYTE g = (GetRValue(c)+GetGValue(c)+GetBValue(c))/3;
        if(g<100) g =255; else g = 0;
        pDC->SetTextColor(RGB(0,g,0));
        pDC->SetBkMode(TRANSPARENT);
        // Display the caption
        pDC->DrawText(info,r,DT_LEFT | DT_END_ELLIPSIS);
        pDC->SelectObject(oldcf);
    }
}

/*****
*
*   Map pixel coordinates from 2D to linear array of pixel bytes
*
*****/
inline unsigned long Image::MapPixel (unsigned int x, unsigned int y)
{
    /* Safe pixel mapping - important ! */
    if(x >= (unsigned int)m_Width) x=m_Width-1;
    if(y >= (unsigned int)m_Height) y=m_Height-1;

    /* Metheus vertical flip */
    if(theApp.app_Metheus) y=m_Height-1-y;

    return m_Bytes_per_Pixel*(m_Width*(long)(m_Height-y-1)+x);
}

/*****
*
*   Converts DIB to DDB
*
*****/
HBITMAP Image::DIB_to_DDB(CDC *pDC)
{
    return CreateDIBitmap(pDC->GetSafeHdc(),&(m_bmpInfo->bmiHeader),
        CBM_INIT,m_Pixels,m_bmpInfo,DIB_RGB_COLORS);
}

```

```

/*****
 *
 * Returns subimage of the current image
 *
 *****/
void Image::GetSubimage(Image* sub, const int xleft, const int ytop)
{
    if(!sub) return;
    // Backup pixel values
    sub->ResetPixels(true);

    int x,y;
    for(x=xleft; x<xleft+(sub->GetWidth()); x++)
    {
        for(y=ytop; y<ytop+(sub->GetHeight()); y++)
        {
            if(x<0 || x>=this->GetWidth() || y<0 || y>=this->GetHeight())
            {
                sub->SetPixel(x-xleft,y-ytop,0); //black background color
            }
            else
            {
                sub->SetPixel(x-xleft,y-ytop,this->GetPixel(x,y));
            }
        }
    }
}

/*****
 *
 * Set image flip parameter
 *
 *****/
void Image::TR_Flip(bool horizontal)
{
    int x, y, w=GetWidth(), h=GetHeight();
    long p;
    if(horizontal) // horizontal flip
    {
        ResetPixels(true); // backup
        for(x=0; x<w/2; x++)
        {
            ::ShowProgress((200*x)/w,"Flipping horizonatllly ..");
            for(y=0; y<h; y++)
            {
                p = GetPixel(x,y);
                SetPixel(x,y,GetPixel(w-x,y));
                SetPixel(w-x,y,p);
            }
        }
    }
    else
    {
        ResetPixels(true); // backup
        for(y=0; y<h/2; y++)
        {
            ::ShowProgress((200*y)/h,"Flipping vertically ..");
            for(x=0; x<w; x++)
            {
                p = GetPixel(x,y);
                SetPixel(x,y,GetPixel(x,h-y));
                SetPixel(x,h-y,p);
            }
        }
    }
    ::ShowProgress(0,"Ready");
    Beep(500,100);
}

```



```

/*****
 *
 *   Increase image brightness by b percent
 *   (decreases brightness for negative b)
 *
 *****/
void Image::SetBrightness(int b)
{
    long amin, amax, bmin, bmax;

    /* Validation */
    if(b<-99) b=-99; else if(b>99) b=99;
    if(b==0) return; // nothing to do;

    /* Find current max and min */
    if(m_pPal->p_active) m_pPal->Get_Pal_minmax(amin,amax);
    else Get_Pixel_minmax(amin,amax);

    /* Set new min and max */
    bmin=(int)(amin+((long)b)*(amax-amin)/100);
    bmax=bmin+(amax-amin);
    if(bmin<0) bmin=0;
    if(bmax>m_maxPixelValue) bmax=m_maxPixelValue;

    /* Remap the pixels */
    TR_HistStretch(amin,amax,bmin,bmax);
    return;
}

/*****
 *
 *   Apply 3x3 Smooth mask:
 *
 *   1   2   1
 *   2   4   2
 *   1   2   1
 *
 *****/
long Image::TR_Smooth(const int x, const int y)
{
    long a00=GetPixel(x-1,y-1);    long a10=GetPixel(x,y-1);    long a20=GetPixel(x+1,y-1);
    long a01=GetPixel(x-1,y);      long a11=GetPixel(x,y);      long a21=GetPixel(x+1,y);
    long a02=GetPixel(x-1,y+1);    long a12=GetPixel(x,y+1);    long a22=GetPixel(x+1,y+1);
    if(!m_RGB)
    {
        long t=(a11<<2)+((a01+a12+a21+a10)<<1)+a02+a22+a00+a20;
        return (t>>4);
    }
    else
    {
        long tb,tg,tr;
        tb=(GetBValue(a11)<<2)
            +((GetBValue(a01)+GetBValue(a12)+GetBValue(a21)+GetBValue(a10))<<1)
            +GetBValue(a02)+GetBValue(a22)+GetBValue(a00)+GetBValue(a20);
        tg=(GetGValue(a11)<<2)
            +((GetGValue(a01)+GetGValue(a12)+GetGValue(a21)+GetGValue(a10))<<1)
            +GetGValue(a02)+GetGValue(a22)+GetGValue(a00)+GetGValue(a20);
        tr=(GetRValue(a11)<<2)
            +((GetRValue(a01)+GetRValue(a12)+GetRValue(a21)+GetRValue(a10))<<1)
            +GetRValue(a02)+GetRValue(a22)+GetRValue(a00)+GetRValue(a20);
        return RGB(min(tr>>4,255),min(tg>>4,255),min(tb>>4,255));
    }
}

/*****
 *
 *   Apply SUSAN-like or 3x3 Gaussian mask to denoise:
 *
 *   1   4   1
 *   4  12   4
 *
 *****/

```

```

*   1   4   1
*
*****/
long Image::TR_DeNoise(const int x, const int y)
{
    static const int    rad=2;
    static const int    rad2=rad*2+1;

    static bool        init_state=false;
    static int          thresh=rad;
    static long         bp[50];
    static long         dpt[rad2][rad2];
    static double        thresh2=1.0/(thresh*thresh);

    long    sigma=50, brightness, ltmp, area, total;

    // Calculate exponent and distance look-up tables
    if (!init_state)
    {
        for(int k=0;k<50;k++)    bp[k]=(long)(100.0*exp(-0.1*k));
        for(int i=-rad; i<=rad; i++)
        {
            for(int j=-rad; j<=rad; j++)
            {
                dpt[i+rad][j+rad] = (long) (100.0 * exp((-i*i-j*j)*thresh2));
            }
        }
        init_state=true;
    }

    // Calculate image squared variance "sigma"
    if((x==rad) && (y==rad))
    {
        GetStatistics(area, sigma, ltmp);
        if(sigma<1) sigma=1;
    }

    // Get central pixel
    long a11=GetPixel(x,y);

    // Do denoising
    if(!m_RGB) // Process grayscale image
    {
        area = 0;
        total = 0;
        for(int k=0; k<rad2; k++)
        {
            for(int l=0; l<rad2; l++)
            {
                brightness=GetPixel(x+k-rad,y+l-rad);
                ltmp=brightness-a11;
                ltmp=(10*ltmp*ltmp)/sigma;
                if (ltmp>=50) continue;
                ltmp=bp[ltmp];
                ltmp *= dpt[k][l];
                area += ltmp;
                total += ltmp * brightness;
            }
        }
        ltmp = area-10000;

        if (ltmp==0)
        {
            return ((a11<<2)+GetPixel(x-1,y)+GetPixel(x+1,y)+
                    GetPixel(x,y-1)+GetPixel(x,y+1))>>3;
        }
        else
            return (total-(a11*10000))/ltmp;
    }
    else// Process color image
    {
        long resultr, resultg, resultb,brl;
        long arear,areag,areab;
        long totalr,totalg,totalb;
    }
}

```

```

arear = areag = areab = 0;
totalr = totalg = totalb = 0;

for(int k=0; k<rad2; k++)
{
    for(int l=0; l<rad2; l++)
    {
        brightness=GetPixel(x+k-rad,y+l-rad);
        //-----red-----
        br1=GetRValue(brightness);
        ltmp=br1-GetRValue(a11);
        ltmp=(10*ltmp*ltmp)/sigma;
        if (ltmp<50)
        {
            ltmp=bp[ltmp];
            ltmp *= dpt[k][l];
            arear += ltmp;
            totalr += ltmp * br1;
        }
        //-----green-----
        br1=GetGValue(brightness);
        ltmp=br1-GetGValue(a11);
        ltmp=(10*ltmp*ltmp)/sigma;
        if (ltmp<50)
        {
            ltmp=bp[ltmp];
            ltmp *= dpt[k][l];
            areag += ltmp;
            totalg += ltmp * br1;
        }
        //-----blue-----
        br1=GetBValue(brightness);
        ltmp=br1-GetBValue(a11);
        ltmp=(10*ltmp*ltmp)/sigma;
        if (ltmp<50)
        {
            ltmp=bp[ltmp];
            ltmp *= dpt[k][l];
            areab += ltmp;
            totalb += ltmp * br1;
        }
    }
}

//-----red-----
ltmp = arear-10000;
if (ltmp==0)
{
    long a00=GetPixel(x-1,y-1);    long a10=GetPixel(x,y-1);
    long a20=GetPixel(x+1,y-1);    long a01=GetPixel(x-1,y);

    long a21=GetPixel(x+1,y);      long a02=GetPixel(x-1,y+1);
    long a12=GetPixel(x,y+1);      long a22=GetPixel(x+1,y+1);
    resultr=(GetRValue(a11)<<3)
        +((GetRValue(a11)+GetRValue(a01)+GetRValue(a12)+GetRValue(a21)+
           GetRValue(a10))<<2)
        +GetRValue(a02)+GetRValue(a22)+GetRValue(a00)+GetRValue(a20);
    if (resultr<0) resultr=0;
    else resultr >= 5;
}
else    resultr = (totalr-(GetRValue(a11)*10000))/ltmp;

//-----green-----
ltmp = areag-10000;
if (ltmp==0)
{
    long a00=GetPixel(x-1,y-1);    long a10=GetPixel(x,y-1);
    long a20=GetPixel(x+1,y-1);    long a01=GetPixel(x-1,y);

    long a21=GetPixel(x+1,y);      long a02=GetPixel(x-1,y+1);
    long a12=GetPixel(x,y+1);      long a22=GetPixel(x+1,y+1);
    resultg=(GetGValue(a11)<<3)
        +((GetGValue(a11)+GetGValue(a01)+GetGValue(a12)+GetGValue(a21)+
           GetGValue(a10))<<2)
        +GetGValue(a02)+GetGValue(a22)+GetGValue(a00)+GetGValue(a20);
    if (resultg<0) resultg=0;
}

```

```

        else resultg >= 5;
    }
    else    resultg = (totalg-(GetGValue(a11)*10000))/ltmp;
    //-----blue-----
    ltmp = areab-10000;
    if (ltmp==0)
    {
        long a00=GetPixel(x-1,y-1);    long a10=GetPixel(x,y-1);
        long a20=GetPixel(x+1,y-1);    long a01=GetPixel(x-1,y);

        long a21=GetPixel(x+1,y);    long a02=GetPixel(x-1,y+1);
        long a12=GetPixel(x,y+1);    long a22=GetPixel(x+1,y+1);
        resultb=(GetBValue(a11)<<3)
            +((GetBValue(a11)+GetBValue(a01)+GetBValue(a12)+GetBValue(a21)+
              GetBValue(a10))<<2)
            +GetBValue(a02)+GetBValue(a22)+GetBValue(a00)+GetBValue(a20);
        if (resultb<0) resultb=0;
        else resultb >= 5;
    }
    else    resultb = (totalb-(GetBValue(a11)*10000))/ltmp;
    //-----total color -----
    //return RGB(min(resultr,255),min(resultg,255),min(resultb,255));
    return resultr;
}
}

```

```

/*
 * Apply 3x3 sharpening mask:
 *
 * -1  -1  -1
 * -1  10  -1
 * -1  -1  -1      /      2
 *
 */
long Image::TR_Sharp(const int x, const int y)
{
    long a00=GetPixel(x-1,y-1);    long a10=GetPixel(x,y-1);    long a20=GetPixel(x+1,y-1);
    long a01=GetPixel(x-1,y);    long a11=GetPixel(x,y);    long a21=GetPixel(x+1,y);
    long a02=GetPixel(x-1,y+1);    long a12=GetPixel(x,y+1);    long a22=GetPixel(x+1,y+1);
    if(!m_RGB)
    {
        long t=((a11<<1)+(a11<<3)-(a01+a12+a21+a10+a02+a22+a00+a20))>>1;
        if(t<0) t=0;
        else if(t>m_maxPixelValue) t=m_maxPixelValue;
        return t;
    }
    else
    {
        long tr,tg,tb;
        tr=( (GetRValue(a11)<<1)+(GetRValue(a11)<<3)
            -(GetRValue(a01)+GetRValue(a12)+GetRValue(a21)+GetRValue(a10)
              +GetRValue(a02)+GetRValue(a22)+GetRValue(a00)+GetRValue(a20)) )>>1;
        if(tr<0) tr=0;
        tg=( (GetGValue(a11)<<1)+(GetGValue(a11)<<3)
            -(GetGValue(a01)+GetGValue(a12)+GetGValue(a21)+GetGValue(a10)
              +GetGValue(a02)+GetGValue(a22)+GetGValue(a00)+GetGValue(a20)) )>>1;
        if(tg<0) tg=0;
        tb=( (GetBValue(a11)<<1)+(GetBValue(a11)<<3)
            -(GetBValue(a01)+GetBValue(a12)+GetBValue(a21)+GetBValue(a10)
              +GetBValue(a02)+GetBValue(a22)+GetBValue(a00)+GetBValue(a20)) )>>1;
        if(tb<0) tb=0;
        return RGB(min(tr,255),min(tg,255),min(tb,255));
    }
}

```

```

/*
 * Apply 3x3 edge detector
 *
 */

```

```

// DCM.h : main header file for the DCM application
//

#ifndef AFX_DCM_H__INCLUDED_
#define AFX_DCM_H__INCLUDED_

#ifdef _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif

// #include <afxdao.h>
#include "resource.h" // main symbols
#include "FileBrowser.h" // Added by ClassView
#include "QUERY\QueryRetrieve.h" // Added by ClassView
#include "QUERY\ODBC.h" // Added by ClassView
// #include "LogFile.h" // Added by ClassView

///// User messages
#define WM_USER_DISPLAY_MOST_RECENT WM_USER+31

////////////////////////////////////
// CDCMApp:
// See DCM.cpp for the implementation of this class
//

class CDCMApp : public CWinApp
{
public:
    bool app_Metheus;
    UINT app_ResolutionScaleFactor;
    int app_SupportedPaletteSize;
    ULONG app_DynamicPaletteStart;
    CString app_DirectoryRoot, app_DirectoryTmp,
    app_DirectoryData;
    CSize app_ScreenResolution;
    CPen app_Pen;
    CFont app_SmallFont, app_MediumFont,
    app_LargeFont;
    RTC app_RTC;
    LogFile app_ClientLog, app_ServerLog;
    //DICOMDatabase app_DataBase;
    ODBCDatabase app_DataBase;
    FileBrowser app_FileBrowser;
    QueryRetrieve app_QueryRetrieve;

    static void DisplayRecords(Array<DICOMRecord> &a, bool from_local);
    void ShowProgress(int percent, char* info=NULL);
    void MoveWindowToCorner(CWnd* wnd, CSize offset=CSize(0,0));
    BOOL TestFunction();
    virtual CDocument* OpenDocumentFile(LPCTSTR lpszFileName)
    { return this->OpenDocumentFile(lpszFileName, true); }
    virtual CDocument* OpenDocumentFile(LPCTSTR lpszFileName, bool AddToDatabase);

    CDCMApp();
    ~CDCMApp();

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CDCMApp)
public:
    virtual BOOL InitInstance();
//}}AFX_VIRTUAL

// Implementation

//{{AFX_MSG(CDCMApp)
afx_msg void OnAppAbout();
afx_msg void OnFileBrowse();

```

```

afx_msg void OnFileQueryRetrieve();
afx_msg void OnUpdateFileQueryRetrieve(CCmdUI* pCmdUI);
afx_msg void OnDataBaseImportFiles();
afx_msg void OnDataBaseAttachFiles();
afx_msg void OnDataBaseRemoveFilesImported();
afx_msg void OnDataBaseAddRecords();
afx_msg void OnDataBaseRemoveRecords();
afx_msg void OnDataBaseRemoveFilesAttached();
afx_msg void OnDataBaseRemoveALLRecords();
//}}AFX_MSG
afx_msg LRESULT OnDisplayMostRecent(WPARAM wParam, LPARAM lParam);
DECLARE_MESSAGE_MAP()

private:
void SerializeApp(bool is_loading);
OProgress app_Progress;

void LoadStdProfileSettings(UINT nMaxMRU = _AFX_MRU_COUNT);
bool TimeBomb();
bool SetDirectories();
};

extern CDCMApp theApp;
extern void ShowProgress(int percent, char* info);
extern CString Trim(char* s);

////////////////////////////////////
// MemoryLeakTest
// Simple Class to test for memory leaks

class MemoryLeakTest
{
public:
#ifdef _DEBUG
private:
CMemoryState m_oldMemState, m_newMemState, m_diffMemState;
#endif

public:
inline void Start()
{
#ifdef _DEBUG
m_oldMemState.Checkpoint();
#endif
return;
};

inline void End(int code=0)
{
#ifdef _DEBUG
m_newMemState.Checkpoint();
if( m_diffMemState.Difference( m_oldMemState, m_newMemState ) )
{
TRACE( "Memory leaked!\n" );
m_diffMemState.DumpStatistics();
CString message;
message.Format("\nLocation Code = %d", code);
if(code) AfxMessageBox( " Memory leaked!" + message );
Beep(900,200); Beep(100,200);
}
else
{
CString message;
message.Format("\nLocation Code = %d", code);
if(code) AfxMessageBox( "No memory leaks" + message );
Beep(100,100); Beep(100,100);
}
#endif
return;
};
};

////////////////////////////////////
// ORecentFileList document

class ORecentFileList : public CRecentFileList
{

```

```

public:
    void    CleanMenuAliases();
    void    AddMenuAlias(LPCTSTR fname, CString mname);
    void    SerializeORFList(FILE* fp, bool is_loading);
    ORecentFileList(UINT nStart, LPCTSTR lpszSection, LPCTSTR lpszEntryFormat,
        int nSize, int nMaxDisplen = AFX_ABBREV_FILENAME_LEN ):
        CRecentFileList(nStart, lpszSection, lpszEntryFormat, nSize, nMaxDisplen)
    {
    };

protected:
    void UpdateMenu(CCmdUI* pCmdUI);
private:
    CMapStringToString m_menuNames;
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous
// line.

#endif // !defined(AFX_DCM_H__INCLUDED_)

```

```

- // DCM.cpp : Defines the class behaviors for the application.
- //
-
#include "stdafx.h"
#include "DCM.h"

#include "MainFrm.h"
#include "ChildFrm.h"
#include "DCMDoc.h"
#include "DCMView.h"
#include <direct.h>
#include "CustomFileDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

void ShowProgress(int percent, char* info)
{
    theApp.ShowProgress(percent, info);
};

CString Trim(char* s)
{
    CString str(s);
    str.TrimLeft(); str.TrimRight();
    return str;
}

///////////////////////////////////////////////////
// CDCMApp
BEGIN_MESSAGE_MAP(CDCMApp, CWinApp)
    //{{AFX_MSG_MAP(CDCMApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    ON_COMMAND(ID_FILE_BROWSE, OnFileBrowse)
    ON_COMMAND(ID_FILE_QUERYRETRIEVE, OnFileQueryRetrieve)
    ON_UPDATE_COMMAND_UI(ID_FILE_QUERYRETRIEVE, OnUpdateFileQueryRetrieve)
    ON_COMMAND(ID_DATABASE_ADDFILES_IMPORT, OnDataBaseImportFiles)
    ON_COMMAND(ID_DATABASE_ADDFILES_ATTACH, OnDataBaseAttachFiles)
    ON_COMMAND(ID_DATABASE_REMOVEFILES, OnDataBaseRemoveFilesImported)
    ON_COMMAND(ID_DATABASE_ADDRECORDS, OnDataBaseAddRecords)
    ON_COMMAND(ID_DATABASE_REMOVERECORDS, OnDataBaseRemoveRecords)
    ON_COMMAND(ID_DATABASE_REMOVEFILES_ATTACHED, OnDataBaseRemoveFilesAttached)
    ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
    ON_COMMAND(ID_DATABASE_REMOVEALLRECORDS, OnDataBaseRemoveALLRecords)
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ///no new files/// ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
    ON_THREAD_MESSAGE(WM_USER_DISPLAY_MOST_RECENT, OnDisplayMostRecent)
END_MESSAGE_MAP()

///////////////////////////////////////////////////
// CDCMApp construction

CDCMApp::CDCMApp()
{
    // TODO: add construction code here
    // Place all significant initialization in InitInstance
    app_DirectoryTmp="";
}

CDCMApp::~CDCMApp()
{
    // Delete graphical objects
    app_Pen.DeleteObject();
    app_SmallFont.DeleteObject();
}

```



```

app_MediumFont.DeleteObject();
app_LargeFont.DeleteObject();

// Clean tmp directory
if( app_DirectoryTmp!="") // Tmp directory exists
{
    WIN32_FIND_DATA wf;
    CString nf=app_DirectoryTmp+"\\*";
    HANDLE hf=FindFirstFile(nf,&wf);
    do
    {
        CString tf=CString(wf.cFileName);
        if(tf!="." && tf!="..")
        {
            DeleteFile(app_DirectoryTmp+"/"+tf);
        }
    }
    while (FindNextFile(hf,&wf));
    FindClose(hf);
}

// Serialize application data
SerializeApp(false);
}
/*****
*
*   Serialize application data
*
*****/
void CDCMApp::SerializeApp(bool is_loading)
{
    // Open data file
    CString fname = "\\app.dat";
    CString f = app_DirectoryData+fname;
    FILE* fp;
    fp = fopen((char*)(LPCSTR)f, is_loading ? "rb" : "wb");
    if(!fp) return;

    // Serialize
    ((ORecentFileList*)m_pRecentFileList)->SerializeORFLList(fp, is_loading);

    // Close data file
    fclose(fp);
}

// The one and only CDCMApp object
CDCMApp theApp;

// CDCMApp initialization
BOOL CDCMApp::InitInstance()
{
    // Ensure that only one DCM copy is running
    HANDLE hMutex = CreateMutex ( NULL, FALSE, "DCM_UNIQUE_MUTEX");
    if ( NULL != hMutex && ERROR_ALREADY_EXISTS == GetLastError() )
    {
        CWnd * hWnd = CWnd::FindWindow( NULL,"DCM_UNIQUE_MUTEX");
        if ( hWnd != NULL )
        {
            // if found make it the current window
            hWnd->SetForegroundWindow();
        }
        AfxMessageBox("DCM is already running !");
        return FALSE;
    }

    // Check the time bomb
    if(!TimeBomb()) return FALSE;

    // Socket support
    if (!AfxSocketInit())

```

```

{
    AfxMessageBox("Windows sockets failed");
    return FALSE;
}

AfxEnableControlContainer();

// Standard initialization
#ifdef _AFXDLL
    Enable3dControls();          // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();    // Call this when linking to MFC statically
#endif

// Change the registry key under which our settings are stored.
// You should modify this string to be something appropriate
// such as the name of your company or organization.
SetRegistryKey(_T("DCM Workstation"));
LoadStdProfileSettings(); // Load standard INI file options (including MRU)

// Register the application's document templates. Document templates
// serve as the connection between documents, frame windows and views.

CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
    IDR_DCMTYPE,
    RUNTIME_CLASS(CDCMDoc),
    RUNTIME_CLASS(CChildFrame), //custom MDI child frame
    RUNTIME_CLASS(CDCMView));
AddDocTemplate(pDocTemplate);

// create main MDI Frame window
CMainFrame* pMainFrame = new CMainFrame; // does not need any "delete" later
if (!pMainFrame->LoadFrame(IDR_MAINFRAME)) return FALSE;
m_pMainWnd = pMainFrame;

// Set application graphics parameters
HDC scrHdc=GetDC(NULL); // grab screen DC GetDC
app_Metheus = (IsMetheusDevice(scrHdc)==TRUE);
app_ScreenResolution.cx=::GetDeviceCaps(scrHdc,HORZRES);
app_ScreenResolution.cy=::GetDeviceCaps(scrHdc,VERTRES);
app_ResolutionScaleFactor=__min(app_ScreenResolution.cx/800,
                                app_ScreenResolution.cx/600);
if(app_ResolutionScaleFactor<1) app_ResolutionScaleFactor=1;
app_Pen.CreatePen(PS_SOLID,2*theApp.app_ResolutionScaleFactor,
    RGB(250,250,250));
int fac=__min(2,theApp.app_ResolutionScaleFactor);
int pix20mm = (25*app_ScreenResolution.cy)/::GetDeviceCaps(scrHdc,VERTSIZE);
app_SmallFont.CreatePointFont(pix20mm, "Arial", NULL);
app_MediumFont.CreatePointFont((3*pix20mm)/2,"Swiss", NULL);
app_LargeFont.CreatePointFont(2*pix20mm, "Swiss", NULL);
if(app_Metheus)
{
    app_SupportedPaletteSize=4096;
    MetheusEscDEVINFO mdinf;
    MetheusGetDEVINFO(scrHdc,&mdinf);
    app_DynamicPaletteStart=mdinf.StartExtraPal+app_SupportedPaletteSize;
}
else
{
    app_DynamicPaletteStart=0;
    if(GetDeviceCaps(scrHdc,RASTERCAPS)&RC_PALETTE)
    {
        app_SupportedPaletteSize=GetDeviceCaps(scrHdc,SIZEPALETTE);
        if(app_SupportedPaletteSize<255) app_SupportedPaletteSize=0;
    }
    else app_SupportedPaletteSize=0;
}
ReleaseDC(NULL,scrHdc);

// Parse command line for standard shell commands, DDE, file open
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);

```

```

// Dispatch commands specified on the command line
if (!ProcessShellCommand(cmdInfo))
    return FALSE;

// The main window has been initialized, so show and update it.
pMainFrame->ShowWindow(m_nCmdShow);
pMainFrame->UpdateWindow();
// Do not initialize an empty document
//pMainFrame->MDIGetActive()->MDIDestroy();
pMainFrame->MDIGetActive()->GetActiveDocument()->OnCloseDocument();

// Set working directories
if(!SetDirectories()) return FALSE;

// Test some code
if(!TestFunction()) return FALSE;

// Initialize File Browser
if(!app_FileBrowser.Initialize(app_DirectoryTmp)) return FALSE;

// Initialize progress control
if(!app_Progress.Initialize()) return FALSE;

// Load RTC
CString path=app_DirectoryRoot;
path=app_DirectoryRoot.Left(app_DirectoryRoot.GetLength()-6); // remove "/Applic"
path+=CString("dcmdict.txt");
bool dictEnabled = (app_RTC.AttachRTC((char*)(LPCSTR)path, ::ShowProgress)==TRUE);
((CMainFrame*)m_pMainWnd)->SetDictionaryStatus(dictEnabled);

// Initialize log files (must go after RTC and directory initialization)
if(!app_ClientLog.CreatedVL(
    (char*)(LPCSTR)(theApp.app_DirectoryData+"\\ClientLog.txt"),&app_RTC))
{
    AfxMessageBox("Cannot initialize client log");
    return FALSE;
}
if(!app_ServerLog.CreatedVL(
    (char*)(LPCSTR)(theApp.app_DirectoryData+"\\ServerLog.txt"),&app_RTC))
{
    AfxMessageBox("Cannot initialize server log");
    return FALSE;
}
if(!app_QueryRetrieve.InitializeQueryRetrieve(&app_ClientLog,
    &app_ServerLog, &app_DataBase))
{
    return FALSE;
}

// Load serialized data
SerializeApp(true);

return TRUE;
}
/*****
*
* Set directory structure
*****/
bool CDCMApp::SetDirectories()
{
    bool newdir=false;
    app_DirectoryTmp="";
    // Get root directory
    app_DirectoryRoot=GetProfileString("Directories","Root","");
    if(app_DirectoryRoot.Find("Profile")>0) app_DirectoryRoot=""; // avoid desktop
    if(app_DirectoryRoot=="") // reset
    {
        newdir=true;
        app_DirectoryRoot=CString(this->m_pszHelpFilePath);
        app_DirectoryRoot.TrimRight();
        app_DirectoryRoot.Replace("/", "\\");
        int n=app_DirectoryRoot.ReverseFind('\\');
        if(n>0) app_DirectoryRoot=app_DirectoryRoot.Left(n);
    }
}

```

```

    app_DirectoryRoot += CString("\\Applic");
    WriteProfileString("Directories","Root",app_DirectoryRoot);
}
// Can we access root directory ?
//if(SetCurrentDirectory(app_DirectoryRoot)==FALSE)
if(!::CreateAndSetCurrentDirectory(
    (char*)(LPCSTR)app_DirectoryRoot))
{
    WriteProfileString("Directories","Root","");
    AfxMessageBox("Failed to setup the application directory");
    return false;
}
// Create subdirectories
app_DirectoryRoot.Replace("/", "\\");
app_DirectoryTmp = app_DirectoryRoot + "\\temp";
app_DirectoryData = app_DirectoryRoot + "\\data";
CreateDirectory(app_DirectoryTmp, NULL);
CreateDirectory(app_DirectoryData, NULL);
CString dbdir = app_DirectoryRoot + "\\DataBase";
if(!app_DataBase.InitializeDataBase((char*)(LPCSTR)dbdir, &DisplayRecords))
{
    WriteProfileString("Directories","Root","");
    return false;
}
if(newdir)
{
    CString info;
    AfxFormatString1(info, IDS_DIRECTORY_SETUP, app_DirectoryRoot);
    AfxMessageBox(info, MB_ICONINFORMATION | MB_OK);
}
return true;
}
*****
* Open (create) a new document
*****
Document* CDCMApp::OpenDocumentFile(LPCTSTR lpszFileName, bool AddToDatabase)
{
    Beep(700,200);
    if(!lpszFileName || lpszFileName=="") return NULL;
    ShowProgress(5,"Parcing image file...");
    CDCMDoc* pDoc = (CDCMDoc*)(CWinApp::OpenDocumentFile(lpszFileName));
    if(!pDoc) return NULL;
    if(pDoc->IsEmpty())
    {
        pDoc->OnCloseDocument(); // kill empty or invalid documents
        ShowProgress(0,"Ready");
        return NULL;
    }
    else // Looks like a nice DICOM document
    {
        ((ORecentFileList*)m_pRecentFileList)->AddMenuAlias(lpszFileName,
            pDoc->GetMRUAlias());
        if(AddToDatabase)
        {
            app_DataBase.DBAdd( *(pDoc->GetDICOMRecordPtr()) );
        }
        ShowProgress(0,"Ready");
        return pDoc;
    }
}

LRESULT CDCMApp::OnDisplayMostRecent(WPARAM wParam, LPARAM lParam)
{
    if(!wParam) return 0L;
    Array<DICOMRecord>* a = (Array<DICOMRecord>*)wParam;
    if(!a) return 0L;
    if(a->GetSize()<=0) return 0L;
    CDCMDoc* pDoc = NULL;

    // Any documents open ?
    POSITION pos = theApp.GetFirstDocTemplatePosition();
    CMultiDocTemplate* pDocTemplate = NULL;

```

```

if(pos)
{
    pDocTemplate = (CMultiDocTemplate*)
        (theApp.GetNextDocTemplate(pos));
}

// Go through all existing documents
if(pDocTemplate)
{
    POSITION posDoc = pDocTemplate->GetFirstDocPosition();
    while(posDoc && a->GetSize()>0)
    {
        pDoc = (CDCMDoc*) (pDocTemplate->GetNextDoc(posDoc));
        if(!pDoc) break;
        pDoc->AddDICOMRecords(*a);
        pDoc->OnViewRefresh();
    }
}

// Open new documents, if needed
while(a->GetSize()>0)
{
    pDoc = (CDCMDoc*) (
        theApp.OpenDocumentFile(a->Get(0).GetFileName(), false));
    a->RemoveAt(0);
    if(!pDoc) continue;
    pDoc->AddDICOMRecords(*a);
    pDoc->OnViewRefresh();
}
return 0L;

void CDCMApp::DisplayRecords(Array<DICOMRecord> &a, bool from_local)

theApp.PostThreadMessage(WM_USER_DISPLAY_MOST_RECENT, (WPARAM) &a,
    (LPARAM) from_local);

while(a.GetSize()>0)    Sleep(100);

*****
Launch DICOM browser
*****
/
void CDCMApp::OnFileBrowse()
{
    if(app_FileBrowser.DoModal()==IDOK)
    {
        this->OpenDocumentFile(app_FileBrowser.m_RequestedFile, true);
    }
}
/*****
*
*   Start or Terminate Query/Retrieve session
*
*****/
void CDCMApp::OnFileQueryRetrieve()
{
    app_QueryRetrieve.SwitchAppearance();
}

void CDCMApp::OnUpdateFileQueryRetrieve(CCmdUI* pCmdUI)
{
    static bool state=false;
    bool check = (app_QueryRetrieve.GetSafeHwnd() != NULL &&
        app_QueryRetrieve.IsIconic() == FALSE);
    pCmdUI->SetCheck(check);
    if(check != state)
    {
        state=check;
        CMainFrame* mf = (CMainFrame*)AfxGetMainWnd();
        if(mf)    mf->EnableLogoAnimation(!check);
    }
}

```

```

    }
}
/*****
*
*   DataBase menu handlers
*
*****/
void CDCMApp::OnDataBaseImportFiles()
{
    // Display modal FileOpen dialog
    CCustomFileDialog fd;
    fd.SetTitle("Select Files/Directories to Import into the Database");
    if (fd.DoModal() != IDOK) return;
    int count = fd.GetSelectedCount();
    if(count<=0) return;
    // Process selected strings
    bool subdir = (fd.m_SelectSubdirectories == TRUE);
    for(UINT ind=0; ind < (UINT)count; ind++)
    {
        ShowProgress(90*(ind+1)/count, "Importing ... ");
        app_DataBase.ImportDirectory((char*)(LPCSTR) (fd.GetSelectedAt(ind)),
                                     subdir);
    }
    ShowProgress(0);
    return;
}

void CDCMApp::OnDataBaseAttachFiles()
{
    // Display modal FileOpen dialog
    CCustomFileDialog fd;
    fd.SetTitle("Select Files/Directories to Attach to the Database");
    if (fd.DoModal() != IDOK) return;
    int count = fd.GetSelectedCount();
    if(count<=0) return;
    // Process selected strings
    bool subdir = (fd.m_SelectSubdirectories == TRUE);
    for(UINT ind=0; ind < (UINT)count; ind++)
    {
        ShowProgress(90*(ind+1)/count, "Attaching ... ");
        app_DataBase.AttachDirectory((char*)(LPCSTR) (fd.GetSelectedAt(ind)),
                                     subdir);
    }
    ShowProgress(0);
    return;
}

void CDCMApp::OnDataBaseRemoveFilesImported()
{
    // Display modal FileOpen dialog
    CCustomFileDialog fd;
    fd.SetTitle("Select Imported Files/Directories to Remove from the Database");
    if (fd.DoModal() != IDOK) return;
    int count = fd.GetSelectedCount();
    if(count<=0) return;
    // Process selected strings
    bool subdir = (fd.m_SelectSubdirectories == TRUE);
    for(UINT ind=0; ind < (UINT)count; ind++)
    {
        ShowProgress(90*(ind+1)/count, "Removing selected ... ");
        app_DataBase.UnImportDirectory(
            (char*)(LPCSTR) (fd.GetSelectedAt(ind)) , subdir);
    }
    ShowProgress(0);
    return;
}

void CDCMApp::OnDataBaseRemoveFilesAttached()
{
    // Display modal FileOpen dialog
    CCustomFileDialog fd;
    fd.SetTitle("Select Attached Files/Directories to Remove from the Database");
    if (fd.DoModal() != IDOK) return;
    int count = fd.GetSelectedCount();
    if(count<=0) return;
    // Process selected strings
    bool subdir = (fd.m_SelectSubdirectories == TRUE);

```

```

for(UINT ind=0; ind < (UINT)count; ind++)
{
    ShowProgress(90*(ind+1)/count, "Removing attached ... ");
    app_DataBase.UnAttachDirectory((char*) (LPCSTR) (fd.GetSelectedAt(ind)),
                                   subdir);
}
ShowProgress(0);
return;
}

void CDCMApp::OnDataBaseAddRecords()
{
    app_QueryRetrieve.DoModeless();
}

void CDCMApp::OnDataBaseRemoveRecords()
{
    DQRSearch ds;
    if(ds.DoModal() != IDOK) return;
    if(AfxMessageBox("Delete specified items from local database ?",
                     MB_YESNO) != IDYES) return;

    DICOMRecord dr;
    ds.WriteIntoDICOMRecord(dr);
    int n = app_DataBase.DBRemove(dr);
    if(n>0)
    {
        CString info;
        info.Format("%d records were removed from the local database\n\n"
                   "Some of the records in the query results window\n"
                   "may no longer be valid!",n);
        AfxMessageBox(info, MB_ICONINFORMATION);
    }
    else AfxMessageBox("No matching records found", MB_ICONINFORMATION);
}

void CDCMApp::OnDataBaseRemoveALLRecords()
{
    app_DataBase.RemoveAllRecords();
}

/*****
Move window to right bottom corner of the screen
(with specified offset)
Use to display utility dialogs
*****/
void CDCMApp::MoveWindowToCorner(CWnd *wnd,
                                CSize offset/*=CSize(0,0)*/)
{
    CRect wr;
    wnd->GetWindowRect(wr);
    wr.OffsetRect(app_ScreenResolution-wr.BottomRight()
                 -CSize(40,40)-offset);
    wnd->MoveWindow(wr,TRUE);
    wnd->BringWindowToTop();
}

/*****
*   Time bomb
*
*****/
bool CDCMApp::TimeBomb()
{
    CTime t = CTime::GetCurrentTime();
    CTime tstop(2000,12,1,1,1,1);
    if(t>=tstop)
    {
        AfxMessageBox("Program expired !", MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    return true;
}

```

```

/*****
*
*   This function is used for code testing only
*
*****/
#include "AccurateTimer.h"
BOOL CDCMApp::TestFunction()
{
/* MemoryLeakTest mlt;
   mlt.Start();
   {
       DicomObject d1, d2;
       d1.LoadFromFile("D:\\\\DICOM\\Data\\0.dcm");
       d2.LoadFromFile("D:\\\\DICOM\\Data\\00.dcm");

       ImageSeries s1, s2;
       s1.ReadDO(d1);
       s2.ReadDO(d2);
       s1.Append(s2);
   }
   mlt.End(1);
   return FALSE;
*/

   return TRUE;
}
/*****
*
*   Displaying progress in the min frame status bar
*****/
void CDCMApp::ShowProgress(int percent, char* info /* =NULL */)
{
   app_Progress.ShowProgress(percent, info);
}
/*****
*
*   Displaying meaningful recent file list names
*****/
void CDCMApp::LoadStdProfileSettings(UINT nMaxMRU /* = _AFX_MRU_COUNT */)
{
   const TCHAR _afxFileSection[] = _T("Recent File List");
   const TCHAR _afxFileEntry[] = _T("File%d");
   const TCHAR _afxPreviewSection[] = _T("Settings");
   const TCHAR _afxPreviewEntry[] = _T("PreviewPages");
   ASSERT_VALID(this);
   ASSERT(m_pRecentFileList == NULL);

   if (nMaxMRU != 0)
   {
       // create file MRU since nMaxMRU not zero
       m_pRecentFileList = new ORecentFileList(0, _afxFileSection, _afxFileEntry,
                                                nMaxMRU);
       m_pRecentFileList->ReadList();
   }
   // 0 by default means not set
   m_nNumPreviewPages = GetProfileInt(_afxPreviewSection, _afxPreviewEntry, 0);
}

////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
   CAboutDlg();

// Dialog Data
//{{AFX_DATA(CAboutDlg)
enum { IDD = IDD_ABOUTBOX };
//}}AFX_DATA

```



```

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CAboutDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
    // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CDCMApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

////////////////////////////////////
// ORecentFileList
void ORecentFileList::UpdateMenu(CCmdUI *pCmdUI)
{
    ASSERT(m_arrNames != NULL);

    CMenu* pMenu = pCmdUI->m_pMenu;
    if (m_strOriginal.IsEmpty() && pMenu != NULL)
        pMenu->GetMenuString(pCmdUI->m_nID, m_strOriginal, MF_BYCOMMAND);

    if (m_arrNames[0].IsEmpty())
    {
        // no MRU files
        if (!m_strOriginal.IsEmpty())
            pCmdUI->SetText(m_strOriginal);
        pCmdUI->Enable(FALSE);
        return;
    }

    if (pCmdUI->m_pMenu == NULL)
        return;

    for (int iMRU = 0; iMRU < m_nSize; iMRU++)
        pCmdUI->m_pMenu->DeleteMenu(pCmdUI->m_nID + iMRU, MF_BYCOMMAND);

    TCHAR szCurDir[_MAX_PATH];
    GetCurrentDirectory(_MAX_PATH, szCurDir);
    int nCurDir = lstrlen(szCurDir);
    ASSERT(nCurDir >= 0);
    szCurDir[nCurDir] = '\\';
    szCurDir[++nCurDir] = '\0';
}

```

```

CString strName;
CString strTemp;
CString strMenu="Ku-ku";
for (iMRU = 0; iMRU < m_nSize; iMRU++)
{
    if (!GetDisplayName(strName, iMRU, szCurDir, nCurDir))
        break;

    // double up any '&' characters so they are not underlined
    LPCTSTR lpszSrc = strName;
    LPTSTR lpszDest = strTemp.GetBuffer(strName.GetLength()*2);
    while (*lpszSrc != 0)
    {
        if (*lpszSrc == '&')
            *lpszDest++ = '&';
        if (_istlead(*lpszSrc))
            *lpszDest++ = *lpszSrc++;
        *lpszDest++ = *lpszSrc++;
    }
    *lpszDest = 0;
    strTemp.ReleaseBuffer();
    // Modification: find menu alias
    if(m_menuNames.Lookup(m_arrNames[iMRU], strMenu) == FALSE)
    {
        strMenu=strTemp;
    }

    // insert mnemonic + the file name
    TCHAR buf[10];
    wprintf(buf, _T("%&d "), (iMRU+1+m_nStart) % 10);
    pCmdUI->m_pMenu->InsertMenu(pCmdUI->m_nIndex++,
        MF_STRING | MF_BYPOSITION, pCmdUI->m_nID++,
        CString(buf) + strMenu);
}

// update end menu count
pCmdUI->m_nIndex--; // point to last menu added
pCmdUI->m_nIndexMax = pCmdUI->m_pMenu->GetMenuItemCount();

pCmdUI->m_bEnableChanged = TRUE;    // all the added items are enabled
*****

Insert "mname" as a menu alias to existing "fname" file
*****/
void ORecentFileList::AddMenuAlias(LPCTSTR fname, CString mname)
{
    m_menuNames.SetAt(fname, mname);
    CleanMenuAliases();
}

/*****
*
*   Remove old menu aliases
*
*****/
void ORecentFileList::CleanMenuAliases()
{
    bool        found;
    int         i;
    POSITION     pos;
    CString     key, val;
    for( pos = m_menuNames.GetStartPosition(); pos != NULL; )
    {
        m_menuNames.GetNextAssoc(pos, key, val);
        found=false;
        for(i=0; i<m_nSize; i++)
        {
            if(m_arrNames[i]==key) {    found = true;    break;    }
        }
        if(!found)    m_menuNames.RemoveKey(key);
    }
}

```

```

}

/*****
*
*   Serialize MRU list aliases
*
*****/
void ORecentFileList::SerializeORFList(FILE* fp, bool is_loading)
{
    if(!fp) return;
    int    n, k;
    char    key[MAX_PATH+1], val[MAX_PATH+1];
    if(is_loading)
    {
        m_menuNames.RemoveAll();
        ::SerializeInteger(fp,n,true);
        if(n<=0) return;
        for(k=0; k<n; k++)
        {
            ::SerializeString(fp, key, true);
            ::SerializeString(fp, val, true);
            m_menuNames.SetAt(key,val);
        }
    }
    else
    {
        CleanMenuAliases();
        n=m_menuNames.GetCount();
        ::SerializeInteger(fp,n,false);
        POSITION    pos;
        CString    ks, vs;
        for( pos = m_menuNames.GetStartPosition(); pos != NULL; )
        {
            m_menuNames.GetNextAssoc(pos, ks, vs);
            sprintf(key,"%s",ks);
            ::SerializeString(fp, key, false);
            sprintf(val,"%s",vs);
            ::SerializeString(fp, val, false);
        }
    }
}

```

```

// ChildFrm.h : interface of the CChildFrame class
//
/////////////////////////////////////////////////////////////////

#ifdef AFX_CHILDFRM_H__039FD0CD_68EA_11D2_9576_00105A21774F__INCLUDED_
#define AFX_CHILDFRM_H__039FD0CD_68EA_11D2_9576_00105A21774F__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CChildFrame : public CMDIChildWnd
{
    DECLARE_DYNCREATE(CChildFrame)
public:
    CChildFrame();

    // Attributes
public:

    // Operations
public:

    // Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CChildFrame)
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void ActivateFrame(int nCmdShow = -1);
    //}}AFX_VIRTUAL

    // Implementation
public:
    virtual ~CChildFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
protected:
    //{{AFX_MSG(CChildFrame)
    afx_msg void OnMove(int x, int y);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
}

/////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous
// line.

#endif // !defined(AFX_CHILDFRM_H__039FD0CD_68EA_11D2_9576_00105A21774F__INCLUDED_)

```

```

// ChildFrm.cpp : implementation of the CChildFrame class
//

#include "stdafx.h"
#include "DCM.h"
#include "DCMView.h"

#include "ChildFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CChildFrame

IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)

BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
    //{{AFX_MSG_MAP(CChildFrame)
    ON_WM_MOVE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CChildFrame construction/destruction

CChildFrame::CChildFrame()
{
    // TODO: add member initialization code here
}

CChildFrame::~CChildFrame()
{
}

BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs
    return CMDIChildWnd::PreCreateWindow(cs);
}

////////////////////////////////////
// CChildFrame diagnostics

#ifdef _DEBUG
void CChildFrame::AssertValid() const
{
    CMDIChildWnd::AssertValid();
}

void CChildFrame::Dump(CDumpContext& dc) const
{
    CMDIChildWnd::Dump(dc);
}

#endif // _DEBUG

////////////////////////////////////
// CChildFrame message handlers

void CChildFrame::ActivateFrame(int nCmdShow)
{
    CMDIChildWnd::ActivateFrame(SW_SHOWMAXIMIZED);
}

/*****
*
*   Responds to view window moves
*****/

```

```

*
*****/
void CChildFrame::OnMove(int x, int y)
{
    CDCMView *pView = (CDCMView*)GetActiveView();
    if(!pView) return;
    pView->EraseTools();

    /*
    // determine desired size of the view
    CRect temp(0,0,0,0);
    pView->CalcWindowRect(temp, CWnd::adjustBorder);
    CalcWindowRect(temp, CWnd::adjustBorder);
    CWnd* pFrameParent = GetParent();

    CRect rcBound;
    pFrameParent->GetClientRect(rcBound);
    //CWinRect rcBound(pFrameParent, CWinRect::CLIENT);

    CRect rectView(CPoint(0, 0), pView->GetTotalSize());
    rectView.right = __min(rectView.right, rcBound.right - temp.Width());
    rectView.bottom = __min(rectView.bottom, rcBound.bottom - temp.Height());
    pView->CalcWindowRect(rectView, CWnd::adjustOutside);
    CalcWindowRect(rectView, CWnd::adjustBorder);
    rectView -= rectView.TopLeft();
    rectView.right = __min(rectView.right, rcBound.right);
    rectView.bottom = __min(rectView.bottom, rcBound.bottom);
    */
}

```

```

// DCMDoc.h : interface of the CDCMDoc class
//
/////////////////////////////////////////////////////////////////

#ifdef _MSC_VER
#define AFX_DCMDOC_H__039FD0CF_68EA_11D2_9576_00105A21774F__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "MainFrm.h"
#include "Lupa.h" // Added by ClassView
#include "DICOM_H\\DICOMDocument.h" // Added by ClassView

class CDCMDoc : public CDocument, public DICOMDocument
{
protected: // create from serialization only
    CDCMDoc();
    DECLARE_DYNCREATE(CDCMDoc)

public:
    double          m_imageZoom, m_imageZoomFitScreen;
    Lupa            m_Lupa;
    CMainFrame*     m_MainFrame;

    bool            OnZoom(UINT nID, Image *pBmp);
    Image*          GotoImage(int code);
    virtual         ~CDCMDoc();

private:
    CString         m_SoundFileName;

    Image*          GetDIBImage(int n);
    Image*          GetDIBImage();
    CView*          GetViewPtr();

public:
    void OnViewRefresh(BOOL erase_background=FALSE);
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CDCMDoc)
public:
    virtual void Serialize(CArchive& ar);
    //}}AFX_VIRTUAL

#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
    //{{AFX_MSG(CDCMDoc)
    afx_msg void OnFileSendMail();
    afx_msg void OnUpdateFileSendMail(CCmdUI* pCmdUI);
    afx_msg void OnUpdateZoomFitScreen(CCmdUI* pCmdUI);
    afx_msg void OnUpdateFrameNumber(CCmdUI* pCmdUI);
    afx_msg void SetFitScreenZoom();
    afx_msg void OnViewDicomInfo();
    afx_msg void OnFileSave();
    afx_msg void OnFileSaveAs();
    afx_msg void OnSoundRecord();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous
line.

```

```
#endif // !defined(AFX_DCMDOC_H__039FD0CF_68EA_11D2_9576_00105A21774F__INCLUDED_)
```



```

// DCMDoc.cpp : implementation of the CDCMDoc class
//

#include "stdafx.h"
#include "DCM.h"

#include "DCMDoc.h"
#include "DCMView.h"
#include "SoundDialog.h"
#include "Tools/Email.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CDCMDoc

IMPLEMENT_DYNCREATE(CDCMDoc, CDocument)

BEGIN_MESSAGE_MAP(CDCMDoc, CDocument)
    //{{AFX_MSG_MAP(CDCMDoc)
    ON_UPDATE_COMMAND_UI(ID_ZOOM_FIT_SCREEN, OnUpdateZoomFitScreen)
    ON_COMMAND(ID_ZOOM_FIT_SCREEN, SetFitScreenZoom)
    ON_COMMAND(ID_VIEW_DICOMINFO, OnViewDicomInfo)
    ON_COMMAND(ID_FILE_SAVE, OnFileSave)
    ON_COMMAND(ID_FILE_SAVE_AS, OnFileSaveAs)
    ON_COMMAND(ID_SOUND_RECORD, OnSoundRecord)
    ON_UPDATE_COMMAND_UI(ID_FRAME_NUMBER, OnUpdateFrameNumber)
    //}}AFX_MSG_MAP
    ON_COMMAND(ID_FILE_SEND_MAIL, OnFileSendMail)
    ON_UPDATE_COMMAND_UI(ID_FILE_SEND_MAIL, OnUpdateFileSendMail)
END_MESSAGE_MAP()

//
// *****
//
// Constructor & Destructor
// *****/
CDCMDoc::CDCMDoc()
{
    m_MainFrame=(CMainFrame*)AfxGetMainWnd();
    m_imageZoomFitScreen=m_imageZoom=1.0; // initial zoom - normal size;

    /* No sound file */
    m_SoundFileName=" ";
}

CDCMDoc::~CDCMDoc() { ; }

// *****
//
// CDCMDoc serilization
// *****/
void CDCMDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        CFile* pFile = ar.GetFile();
        if(!pFile)
        {
            AfxMessageBox("Cannot open a file");
            return;
        }
        DICOMDocument::SaveDocument(pFile->GetFilePath(),
                                    DICOMDocument::FormatDICOMModified);
        return;
    }
    else

```

```

{
    CFile* pFile = ar.GetFile();
    if(!pFile)
    {
        AfxMessageBox("Cannot open a file");
        return;
    }
    ar.Flush(); // flush all data into the file
    // Can we load the DICOM ?
    if(!DICOMDocument::LoadFile(pFile->GetFilePath())) return;
    //AddDDOFile("D:\\Dicom\\Data\\0.dcm"); // test
    // Do we have any images ?
    if(GetNumberOfImages()<=0)
    {
        AfxMessageBox("This DICOM object contains no image data",
            MB_OK|MB_ICONINFORMATION);
        return;
    }
    // Update image view
    UpdateAllViews(NULL, 0, NULL );
    return;
}

/*****
 *
 *   DEBUG diagnostics
 *****/
#ifdef _DEBUG
void CDCMDoc::AssertValid() const
{
    CDocument::AssertValid();
}
void CDCMDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

/*****
 *
 *   Get image #n or current frame
 *****/
Image* CDCMDoc::GetDIBImage(int n)
{
    Image* img = DICOMDocument::GetImage(GetCurrentImageIndex());
    if(!img) return NULL;
    if(img)
    {
        m_imageZoom = img->GetZoom();
    }
    img = DICOMDocument::GetImage(n);
    if(img)
    {
        m_MainFrame->ShowFrameNumber(GetCurrentImageIndex()+1);
        img->m_ScreenMap.ValidateZoom(m_imageZoom); // zoom appropriately
    }
    GetCurrentSeries()->SetSelectedImage();
    return img;
}

Image* CDCMDoc::GetDIBImage()
{
    return GetDIBImage(DICOMDocument::GetCurrentImageIndex());
}

/*****
 *
 *   Go to first (code=-2), previous (code=-1),
 *   next (code=+1) or last (code=+2) image
 *****/

```

```

*
*****/
Image* CDCMDoc::GotoImage(int code)
{
    switch(code)
    {
        case -2:    return GetDIBImage(0);
        case -1:    return GetDIBImage(GetCurrentImageIndex()-1);
        case 1:     return GetDIBImage(GetCurrentImageIndex()+1);
        case 2:     return GetDIBImage(GetNumberOfImages()-1);
    }
    return GetDIBImage(GetCurrentImageIndex());
}

/*****
*
*   Save DICOM document in different formats
*
*****/
void CDCMDoc::OnFileSaveAs()
{
    BYTE    format;
    CString ffilter=_T( "DICOM (*) |*|"
        "Windows directory (*.bmp, *.txt, *.wav) |*.||");
    CString fname=DICOMDocument::GetFilename();

    // Display modal file dialog
    CFileDialog fd( FALSE, NULL, fname,
        OFN_HIDEREADONLY|OFN_NONETWORKBUTTON|OFN_OVERWRITEPROMPT,
        ffilter);
    if (fd.DoModal() != IDOK ) return;

    // Get save
    fname=fd.GetPathName();
    switch(fd.m_ofn.nFilterIndex)
    {
        case 1:     format=DICOMDocument::FormatDICOMModified;    break;
        case 2:     format=DICOMDocument::FormatWINDOWSMultimedia; break;
        default:    format=DICOMDocument::FormatDICOMOriginal;    break;
    }
    DICOMDocument::SaveDocument(fname,format);
    UpdateAllViews(NULL);
}

void CDCMDoc::OnFileSave()
{
    DICOMDocument::SaveDICOM();
    UpdateAllViews(NULL);
}

/*****
*
*   Reset Doc image zoom; return true if reset was possible
*
*****/
bool CDCMDoc::OnZoom(UINT nID, Image *pBmp)
{
    switch(nID)
    {
        case ID_ZOOM_1_1: m_imageZoom=1.0;    break;
        case ID_ZOOM_1_2: m_imageZoom=2.0;    break;
        case ID_ZOOM_1_3: m_imageZoom=3.0;    break;
        case ID_ZOOM_1_4: m_imageZoom=4.0;    break;
        case ID_ZOOM_2_1: m_imageZoom=0.5;    break;
        case ID_ZOOM_3_1: m_imageZoom=1.0/3;   break;
        case ID_ZOOM_4_1: m_imageZoom=0.25;   break;
        case 1: // next smaller zoom
            if(m_imageZoom<0.2) return false;
            m_imageZoom /= 1.1;
            break;
        case 2: // next larger zoom
            if(m_imageZoom>4) return false;
            m_imageZoom *= 1.1;
            break;
    }
}

```

```

default: return false; // kick out invalid zoom maps
}
pBmp->m_ScreenMap.ValidateZoom(m_imageZoom);
m_Lupa.Initialize(m_Lupa.l_scnSize, m_imageZoom, m_Lupa.l_zoom);
return true;
}
/*****
*
*   Set zoom to fit the entire application screen
*
*****/
void CDCMDoc::SetFitScreenZoom()
{
    if(GetNumberOfImages()<=0) return;
    int img_num=GetCurrentImageIndex(); // save current image index
    Image *img = DICOMDocument::GetImage(0);
    CRect main_client;
    AfxGetMainWnd()->GetClientRect(main_client);
    double screen_width=main_client.Width()-16;
    double screen_height=main_client.Height()-100;
    if(screen_width<16 && screen_height<16) return;
    m_imageZoomFitScreen=min(screen_width/img->GetWidth(),
                             screen_height/img->GetHeight());
    if(m_imageZoomFitScreen>4.0) m_imageZoomFitScreen=4.0;
    m_imageZoom=m_imageZoomFitScreen;

    img->m_ScreenMap.ValidateZoom(m_imageZoom);
    m_Lupa.Initialize(m_Lupa.l_scnSize, m_imageZoom, m_Lupa.l_zoom);

    int off_x=(int)(screen_width-m_imageZoomFitScreen*img->GetWidth())/2;
    int off_y=(int)(screen_height-m_imageZoomFitScreen*img->GetHeight())/2;

    // Reset all frames, browse included
    for(int n=0; n<GetNumberOfImages(); n++)
    {
        DICOMDocument::GetImage(n)->m_ScreenMap.SetLeftTopScreenPoint(off_x+4,off_y+4);
    }
    // Reset to original current image
    DICOMDocument::GetImage(img_num);
    UpdateAllViews(NULL);
}

void CDCMDoc::OnUpdateZoomFitScreen(CCmdUI* pCmdUI)
{
    CString zinfo;
    zinfo.Format("Fit view: %.0lf%%",100*m_imageZoomFitScreen);
    pCmdUI->SetText(zinfo);
    pCmdUI->Enable();
}

/*****
*
*   Launch Sound Recorder
*
*****/
void CDCMDoc::OnSoundRecord()
{
    SoundDialog sd;
    sd.DoModal();
    m_SoundFileName=sd.GetWavFileName();
}

/*****
*
*   Email document as attachment
*
*****/
void CDCMDoc::OnFileSendMail()
{
    CDocument::OnFileSendMail();
    /*
    Email mail;

```

```

CString message = CString("\nAttached is DICOM image \n");
mail.Send("", "DICOM image", message, //DICOMDocument::GetFilename()
"", AfxGetMainWnd()->GetSafeHwnd());
*/
}
void CDCMDoc::OnUpdateFileSendMail(CCmdUI* pCmdUI)
{
    CDocument::OnUpdateFileSendMail(pCmdUI);
}
/*****
*
*   Display DICOM header info
*
*****/
void CDCMDoc::OnViewDicomInfo()
{
    DICOMDocument::DisplayDICOMInfo();
}

/*****
*
*   Update toolbar image counter
*
*****/
void CDCMDoc::OnUpdateFrameNumber(CCmdUI* pCmdUI)
{
    m_MainFrame->ShowFrameNumber(GetCurrentImageIndex()+1);
}

/*****
*
*   Get pointer to the current view
*
*****/
CView* CDCMDoc::GetViewPtr()
{
    POSITION pos = GetFirstViewPosition();
    if(pos != NULL) return GetNextView(pos);
    else return NULL;
}

/*****
*
*   Redraw current view
*
*****/
void CDCMDoc::OnViewRefresh(BOOL erase_background/*=FALSE*/)
{
    CDCMView* pView = (CDCMView*)GetViewPtr();
    if(!pView) return;
    pView->OnViewRefresh(erase_background);
}

```

```

// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_MAINFRM_H__INCLUDED_
#define AFX_MAINFRM_H__INCLUDED_

#ifdef _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMainFrame : public CMDIFrameWnd
{
    DECLARE_DYNAMIC(CMainFrame)
public:
    CMainFrame();

// Attributes
public:

// Operations
public:

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CMainFrame)
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual BOOL DestroyWindow();
//}}AFX_VIRTUAL

// Implementation
public:
    void SetDictionaryStatus(bool enabled);
    void EnableLogoAnimation(bool enable);
    void ShowMultiframeToolbar(bool show=true);
    void ShowFrameNumber(int n);
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected: // control bar embedded members
    CStatusBar m_StatusBar;
    IEToolBar m_ToolBarBasic, m_ToolBarMultiframe;
    CAnimateCtrl m_Animate;
    CReBar m_ReBar;
    CDialogBar m_wndDlgBar;

// Generated message map functions
protected:
   //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnViewToolbar();
    afx_msg void OnUpdateViewToolbar(CCmdUI* pCmdUI);
    afx_msg void OnDropFiles(HDROP hDropInfo);
   //}}AFX_MSG
    afx_msg void OnUpdateProgressStatus(CCmdUI *pCmdUI);
    afx_msg void OnToolbarDropDown(NMTOOLBAR* pnmh, LRESULT* plRes);
    DECLARE_MESSAGE_MAP()
private:
    bool m_showToolbars;
    CString m_paneString;
};

/////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous
// line.

#endif // !defined(AFX_MAINFRM_H__INCLUDED_)

```